Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Univerzitetni študij

Diplomska naloga

# Uporaba odprte kode kot osnova za razvoj programske opreme

Peter Primožič

Mentor:
prof. dr. Franc Solina, univ. dipl. ing.

Ljubljana, februar 2005

# Kazalo

# Slike

# Tabele

# POVZETEK

Namen diplomske naloge je predstaviti enega od načinov, kako lahko podjetje, ki se ukvarja z razvojem programske opreme, izkoristi fenomen, imenovan odprta koda, za znižanje stroškov in pospešitev razvoja programske opreme. Namesto razvoja željene aplikacije od samega začetka, se podjetjem danes ponuja možnost, da poiščejo podobno rešitev med odprtokodno programsko opremo, ki jo lahko prilagodijo in nadgradijo, da ustreza njihovim zahtevam in potrebam.

Predstavljen je fenomen odprte kode, raziskani in podani so možni načini njene uporabe pri razvoju programske opreme. Podrobno je prikazan primer uporabe odprtokodnega programa v podjetju Comland d.o.o. Podani so razlogi in okoliščine odločitve za uporabo programa ter postopek njegove prilagoditve in razširitve potrebam podjetja. Predstavljeni so končni rezultati preoblikovanja programa in ugotovitve, na podlagi izkušnje v podjetju.

# 1 UVOD

Zamisel o izbrani temi diplomske naloge se mi je porodila ob razvoju aplikacije za uporabniško podporo v podjetju Comland d.o.o., kjer trenutno opravljam študentsko delo. Splet okoliščin nas je v podjetju pripeljal do odločitve, da bo razvoj naše aplikacije temeljil na osnovi odprtokodne rešitve oziroma produkta. Z odprto kodo takrat še nisem bil dobro seznanjen. Vedel sem le, da obstaja in da so odprtokodni programi posebne vrste programi. Lahko rečem, da je nisem jemal preveč resno in se na splošno nisem zavedal njenega pomena ter vse večje uveljavljenosti na področju razvoja programske opreme. Podobnih misli so bili tudi moji sodelavci, zato se mi je zdela naša odločitev o uporabi odprtokodnega produkta zelo zanimiva in na nek način tvegana. Začel sem prebirati članke in publikacije na temo odprte kode; njena ideja in filozofija sta mi postajali vedno bolj všeč. Tako sem se odločil ta fenomen še bolj raziskati in ga predstaviti v okviru diplomske naloge kot teoretično ozadje osrednjega dela diplome. To je predstavitev izdelave aplikacije za podporo uporabnikom storitev podjetja, katere razvoj temelji na osnovi odprtokodne aplikacije za uporabniško podporo (podajanje prijav napak, predlogov za izboljšave ter nadgradnje v zvezi s programsko in strojno opremo).

Značilnosti odprte kode so njena svobodnost (t.j. dostopnost izvorne kode programa, možnost prilagajanja in nadaljnja distribucija spremenjene izvorne kode), podpora odprtih standardov in protokolov ter hitro rastoča množica uporabnikov in s tem posredno tudi razvijalcev.
Predpogoj za uresničevanje ideje zagovornikov odprte kode in povečanje njene popularnosti je bil prav gotovo pojav spleta(interneta). Pred tem je zares živela le med zanesenjaki v laboratorijih, raziskovalnih ustanovah in raznih skupnostih. Z enostavnim dostopom do spleta pa se je možnost deljenja programov in skupnega razvoja programske opreme razvijalcev s celega sveta približala široki množici ljudi. Od takrat naprej ima odprta koda vse opaznejšo vlogo v kolesju svetovne informacijske industrije. Kljub temu odprto kodo danes še vedno mnogi poskušajo odpraviti z zamahom roke in komentarjem, češ da gre za slabo napisane programe, ki jih uporabljajo hekerji in zastonjkarji [30]. Pri tem prednjačijo predvsem velike multinacionalke, ki vidijo v odprti kodi velikega konkurenta in jo skušajo na vsak način zrušiti. Trenutno zelo aktualen primer je izvajanje pritiska »velikih« za sprejem patentne direktive v evropskem parlamentu, ki bi omogočila patentiranje programskih rešitev. To bi za odprto kodo pomenilo veliko škodo, saj je glede na dostopnost izvorne kode, možnost

detekcije patentnih kršitev precej večja, zato bi se v tem primeru utegnilo število njenih uporabnikov in razvijalcev občutno zmanjšati. Na drugi strani obstajajo tudi izjeme med velikimi multinacionalkami, ki so v odprti kodi videle svojo priložnost. Tako je na primer IBM v odprto kodo (predvsem operacijski sistem Linux) vložil precej svoje tehnologije z namenom, da bi se le-ta bolj široko uporabljala, s čimer bi popularizirali druge svoje izdelke, ki uporabljajo enake tehnologije.

Diplomska naloga je sestavljena iz dveh delov. V prvem delu najprej predstavim zgodovino odprte kode, kjer opišem, kako je ideja deljenja izvorne kode programov sploh zaživela, kako se je razvijala skozi čas, kdo so bili njeni pobudniki in utemeljitelji. Nadaljujem z definicijo prostega programja in definicijo odprte kode, kjer predstavim pravne, moralne, etične in filozofske temelje ter pomen obeh izrazov. Sledi kratka predstavitev glavnih značilnosti najbolj popularnih licenc (zakonitih dokumentov), na podlagi katerih je grajeno prosto oziroma odprtokodno programje. Nato se osredotočim na odprtokodne skupnosti, v katerih se razvija prosto programje. V okviru tega poglavja odgovorim na vprašanja, kdo so ljudje, ki razvijajo prosto programje, zakaj to počno oziroma kakšni so njihovi motivi. Sledi poglavje o odprtokodnih projektih, to je projektih razvoja prostega programja znotraj odprtokodnih skupnosti. V tem poglavju predstavim razvojni model odprtokodnih projektov, razložim kdo so uporabniki prostega programja in kakšne so lahko njihove vloge pri razvoju programja. Prav tako predstavim tipičen razvojni cikel odprtokodnih projektov. Prvi del diplomske naloge nadaljujem s predstavitvijo prednosti in slabosti prostega programja, predvsem s stališča končnih uporabnikov. V nadaljevanju se nekoliko dotaknem še ekonomskih vidikov prostega programja, kjer opišem nekaj možnih poslovnih modelov za trženje odprtokodnih produktov. Prvi del zaključim s predstavitvijo najbolj tipičnih primerov uporabe prostega programja pri domačih uporabnikih in v podjetjih, ki razvijajo programsko opremo.

V drugem delu diplomske naloge predstavim odprtokodno aplikacijo z imenom Bugzilla in način njene uporabe v zgoraj omenjenem podjetju Comland. V okviru tega predstavim tudi razloge za odločitev za uporabo te aplikacije ter njeno prilagoditev potrebam podjetja.

# 2 FENOMEN ODPRTE KODE

Ideja odprte kode ima svoje začetke v 60-ih letih dvajsetega stoletja. Kot têrmin oziroma izraz pa ga je šele leta 1998 skovala Iniciativa za odprto kodo[1]. Zgodovina odprte kode je tesno povezana z zgodovino UNIX-a[2]. Odprta koda za svoj uspeh potrebuje pravno varstvo na področju avtorskega in pogodbenega prava. Z licencami in podobnimi ukrepi se namreč omogoči izvrševanje njene ideje (npr. obvezno vključevanje izvorne kode ob izdaji programa).

## *2.1 Zgodovina odprte kode*

Ko ljudje govorijo o prostem programju, se največkrat sklicujejo na operacijski sistem GNU/Linux in njegove aplikacije. Linux-ova zgodovina je dolga in sega še pred čas začetka razvoja UNIX-a (1969). To bogato zgodovinsko ozadje je pomembno za razumevanje odprte kode, saj smo lahko večino njenih filozofskih načel prvič srečali prav v kontekstu operacijskega sistema Unix.

Projekti razvoja prostega programja so se prvič pojavili med skupnostmi »hekerjev[3]«, še posebej znotraj skupine programerjev, zaposlenih v Laboratoriju za umetno inteligenco na MIT-u[4] v 60-ih in 70-ih letih prejšnjega stoletja. Programerji so si rutinsko izmenjavali in nadgrajevali programe. V takih »hekerskih« skupnostih je deljenje programja, sodelovalno učenje in odprtost postala stvar vsakdana.

### 2.1.1 GNU projekt

Mnogi mislijo, da je bil začetnik odprte kode Linus Torvalds, tvorec jedra sistema Linux, vendar ni čisto tako. Eden od najpomembnejših tvorcev odprte kode je Richard Stallman, programer na MIT-u. Bil je genij in ideolog. »Programi, ki jih je napisal so vedno vsebovali še sporočilo, v katerem uporabnike poziva, naj njegove programe delijo z drugimi uporabniki, naj se iz njih česa naučijo, naj jih izboljšujejo in, ko z njim končajo, naj svoje znanje delijo naprej«[4]. V tistem času so računalnike najbolj intenzivno uporabljale univerze in njihovi

[1] Open Source Initiative - OSI
[2] operacijski sistem, ustvarjen v AT&T Bell Laboratories
[3] heker je izraz, ki ima danes za večino ljudi negativen prizvok. Izraz heker večina ljudi uporablja za opisovanje računalniških kriminalcev in vsiljivcev, ki vdirajo v tuje sisteme. V programerskih skupnostih pa ima izraz heker drugačen pomen. Z njim se opiše posebej briljantnega programerja ali tehničnega eksperta na kakšnem drugem področju računalništva.
[4] Massachusetts Institute of Tecnology je raziskovalna ustanova in Univerza, ki se nahaja v mestu Cambridge

raziskovalni laboratoriji. V takih okoljih se je računalniške programe uporabljalo predvsem kot raziskovalna orodja, zato je bilo njihovo deljenje med razvijalci običajna praksa [11].

Leta 1971 se je Stallman priključil skupnosti za deljenje programja, ki je obstajala že več let. Stallman je delovanje skupnosti opisal z besedami: »Vsakič, ko so ljudje iz drugih univerz ali podjetij želeli prenesti ali uporabiti nek program, smo jim to z veseljem pustili. Če si videl nekoga uporabljati nepoznan in zanimiv program, si lahko vedno vprašal za izvorno[5] kodo, da bi jo pregledal, po potrebi spremenil ali pa uporabil kakšen njen del za izdelavo novega programa.« [12]

V začetku 1980-ih se je situacija dramatično spremenila. V tistem času so namreč prenehali z izdelovanjem računalniških sistemov serije Digital PDP-10[6], ki so jih v skupnosti za deljenje programja množično uporabljali. Tako so postali vsi njihovi programi neuporabni, saj so bili napisani v zbirnem jeziku[7], računalniki tiste dobe pa so imeli svoje lastniške operacijske sisteme, ki niso bili kompatibilni z njihovimi programi. Podjetje AT&T, ki je imelo v lasti operacijski sistem Unix, je začelo z licenciranjem uporabe Unix-a. Veliko programerjev je iz univerz in raziskovalnih laboratorijev prešlo v zasebna programska podjetja. Trend izdelovanja lastniških programskih paketov in izdajanja programja v obliki, ki ni omogočala popravljanja ali proučevanja izvorne kode je postajal vse bolj razširjen. Kooperativna skupnost tako ni bila več možna. »Vse to je Stallmana prisililo v sprejem moralne odločitve med naslednjimi tremi možnostmi:

1. Lahko bi se pridružil »svetu lastniškega programja« in prenehal z izdelovanjem programov, ki bi bili odprti za vse.
2. Lahko bi prenehal s programiranjem in s tem preprečil zlorabo svojih spretnosti, znanj.
3. Lahko bi iskal način biti programer in delati na osnovanju nove kooperativne skupnosti oziroma odprtega okolja, ki ga je bil vajen«[9].

Stallman je izbral zadnjo možnost. Zavedal se je, da je za dosego cilja, ki si ga je zastavil, najbolj kritična komponenta izdelava odprtega operacijskega sistema, saj brez njega računalnika ni moč uporabljati. Odločil se je, da bo njegov operacijski sistem kompatibilen z Unix-om in brezplačen, ter da bo na voljo tudi izvorna koda vseh programov v njem.

[5] izvorna koda programov je napisana v računalniških jezikih kot so C, Java, Pascal itd. Proizvajalci lastniškega programja običajno distribuirajo samo objektno kodo (zaporedje števil 0 in 1, ki jih razume računalnik).

[6] je kratica za Programmed Data Processor model 10. Proizvajalo ga je podjetje Digital Equipment Corporation (DEC) v poznih 1960-ih letih. Bil je zelo popularen v hekerskih skupnostih, saj so ga v 1970-ih letih adaptirali v številnih raziskovalnih laboratorijih in računalniških delih Univerz.

[7] nižji programski jezik z naborom ukazov določene centralne procesne enote. Zbirnik je prevajalnik, ki pretvori izvorno kodo, napisano v zbirnem jeziku, v strojni jezik.

Poimenoval ga je GNU, kar je rekurzivni akronim, ki pomeni »GNU is NOT UNIX«. S tem je hotel povedati, da ne gre še za eno komercialno različico sistemov Unix, ki jih je bilo takrat na pretek in so bile med seboj praktično nezdružljive. Tako je nastal manifest GNU[8], kjer je Stallman razkril svoje namene in ideje.

Januarja 1984 je opustil delo pri MIT-u in se v celoti posvetil GNU projektu. Na njegovo veselje, se mu je pri uresničevanju njegove ideje pridružila horda programerjev. Kmalu je postalo jasno, da sámo izdelovanje programja za javnost, ne bo služilo primarnemu cilju projekta (t.j. pustiti uporabnikom svobodo), saj je bilo možno programe rahlo prirediti in jih spremeniti v lastniško programje. To je pripeljalo do nastanka GNU splošne javne licence (GPL), ki je preprečila takšno ravnanje.

Leta 1985 je Stallman skupaj z ljudmi, ki so delali na projektu GNU, ustanovil dobrodelno ustanovo, imenovano Free Source Foundation[9] (FSF), za upravljanje s poslovnimi zadevami projekta, kot so donacije, prodajanje kopij prostega programja ter nudenje drugih storitev.

Do leta 1990 je bil operacijski sistem GNU skoraj končan; edina pomembna komponenta, ki je manjkala, je bilo jedro sistema [9].

### 2.1.2 Linux

Na srečo je v tistem času Linus Torvalds, finski univerzitetni študent, začel z razvojem svojega Unix kompatibilnega jedra, ki ga je poimenoval Linux[10]. Torvalds je zaščitil Linux s konceptom »copyleft«[11] in povabil vsakogar, ki bi želel pomagati, k razvoju in izboljševanju jedra ter odpravljanju hroščev[12]. Razvojna skupnost je zelo hitro rastla in napredovanje sistema je bilo hitro. Zahvaljujoč svojemu odprtemu razvojnemu modelu ter vse pomembnejši vlogi interneta, je Torvalds lahko sodeloval s stotinami programerjev. Okrog leta 1992 je bil rezultat povezave Linuxa z nedokončanim sistemom GNU popolnoma brezplačen operacijski sistem. Od takrat so bile izdane močno izboljšane in razširjene verzije Linux jedra in GNU programskih orodij, kar je popularnost operacijskega sistema Linux vsebolj povečevalo [9].

---

[8] v manifestu, ki je izšel marca 1985, je Stallman podal razlage in definicije ciljev GNU projekta. Za večino ljudi gibanja za prosto programje predstavlja manifest ključen filozofski temelj, vir. GNU Manifest je dosegljiv na naslovu: http://www.gnu.org/gnu/manifesto.html

[9] neprofitna organizacija, ustanovljena za podporo gibanja za prosto programje in še posebej GNU projekta. Več o njej si lahko preberete na naslovu: http://www.fsf.org

[10] zanimivo je, da je Linus s projektom Linux začel bolj za šalo kot zares. Na enem od novičarskih seznamov je oznanil, da za hobi razvija svoj operacijski sistem. Menil je, da sistem ne bo nikoli tako mogočen in tako profesionalen kot GNU in je povabil k sodelovanju vse, ki bi bili pripravljeni pomagati pri razvoju in s povratnimi informacijami.

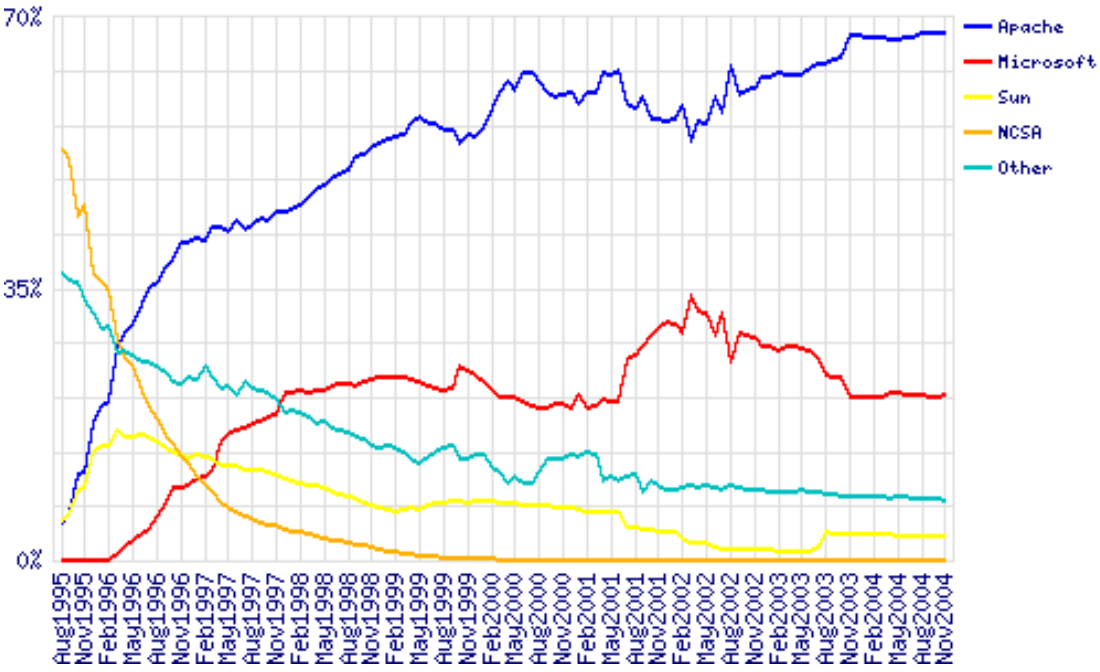
[11] več o tem konceptu si lahko preberete v poglavjih 2.2.1 in 2.2.2

[12] napaka v računalniškem programu, ki povzroči nepravilno oziroma nezaželeno delovanje programa

### 2.1.3 Današnji čas

Leta 1998 so se pri podjetju Netscape[13], ki se je takrat s svojim spletnim brskalnikom Netscape Communicator bojevalo proti Microsoftovemu[14] brskalniku Internet Explorer, odločili za drzno potezo - objavili so izvorno kodo svojega brskalnika. Ta objava je popularnost odprte kode ponesla v višave in mediji so masovno pokazali zanimanje. Septembra 1998 je Microsoft prvič priznal Linux kot konkurenco. Leta 1999 se je odprlo veliko odprtokodnih podjetij in njihove delnice na borzi so spektakularno rastle. V določenih trenutkih je bilo prosto programje označeno kot rešitev, ki bo pozdravila vse bolezni v programskem inženirstvu. V letu 2000 pa so se začele tudi prve težave. Veliko odprtokodnih podjetij je propadlo in oznanilo svoj bankrot, kar je vse skupaj postavilo na realna tla.

Danes projekti odprte kode živijo naprej in vsak dan se pojavljajo novi, njena popularnost je še vedno velika. Poročila o propadu prostega programja so bila tako prenagljena. Dovolj zgovorne so številke, ki jih prikazuje slika 2.1. Po meritvah Netcraft-a[15] je danes skoraj 70% vseh spletnih strežnikov Apache[16] strežnikov. Prav tako je znan podatek, da skoraj tri četrtine spletnih strani temelji na LAMP[17] platformi.




**Slika 2.1 Tržni delež strežnikov na spletu od Avgust 1995 - November 2004**

[13] http://www.netscape.com
[14] http://www.microsoft.com
[15] http://news.netcraft.com
[16] http://www.apache.org
[17] kratica za kombinacijo: Linux, Apache, MySql, (Perl, PHP in/ali Python).

## *2.2 Definicija prostega programja in odprte kode*

Potrebno je razlikovati med odprtokodno programsko opremo in prostim programjem. Gledano s strani programske kode imata oba izraza enak pomen, namreč programska koda mora biti razpoložljiva za vpogled, spreminjanje in nadaljnjo distribucijo.

### 2.2.1 Prosto programje[18]

Prosto programje je izraz, ki ga trdno zagovarja organizacija FSF (Free Software Foundation). Njihovo stališče je, da je prosto programje stvar prostosti in ne cene. Razmišljati je potrebno o tem, da je prosto v pomenu svobode govora, ne brezplačnega piva. Nanaša na prostost uporabnika, da poganja, prepisuje, distribuira, preučuje, spreminja in izboljšuje programje. Natančneje rečeno se nanaša na štiri vrste prostosti, ki uporabnikom programja omogočajo, da lahko:

- prosto poganjajo program za kakršenkoli namen (prostost 0),
- prosto preučujejo delovanje programa in ga prilagajajo svojim potrebam (prostost 1). Predpogoj za to je dostop do izvorne kode,
- prosto razširjajo kopije programa (prostost 2),
- prosto izboljšujejo program in javno izdajajo svoje izboljšave (prostost 3). Predpogoj za to je dostop do izvorne kode.

Programje je prosto samo, če daje uporabnikom vse te pravice. Torej imajo uporabniki pravico izvode programa razširjati z ali brez sprememb, brezplačno ali z zaračunavanjem za razširjanje, komurkoli in kjerkoli, ne da bi za to prosili za dovoljenje. Prav tako je dopustno spreminjati izvode programa in jih uporabljati le za lastne potrebe, ne da bi sploh omenili, da obstajajo. Lahko pa jih tudi objavijo, na da bi kogarkoli na kakršenkoli način obvestili o tem.
Prostost uporabe programa pomeni, da ga lahko katerakoli oseba ali organizacija svobodno uporablja na kakršnemkoli računalniškem sistemu za kakršnokoli delo.
Svoboda redistribucije izvodov mora vključevati binarne ali izvedljive oblike programa in izvorno kodo tako spremenjenih kot nespremenjenih različic programa. Če ni moč narediti binarne ali izvedljive oblike, pa morajo posamezniki vsaj imeti svobodo, da razširjajo takšne oblike, če najdejo ali razvijejo način za njihovo izdelavo. Da bi imela prostost spreminjanja

[18] povzeto po [31]

programa in objavljanja izboljšanih različic smisel, je potrebno imeti dostop do izvirne kode programa. Dostopnost izvorne kode je torej potreben predpogoj za prosto programje.

»Vse te svoboščine uporabnikov prostega programja morajo biti nepreklicne – prvotni imetnik prostega programa (npr. programer) jih ne more naknadno preklicati ali omejiti (razen, če uporabnik krši pravila proste licence)« [7].

Določena pravila in omejitve (npr. v licenčni pogodbi) glede načina distribuiranja prostega programja so sprejemljive, dokler te ne ovirajo osrednjih svoboščin. Primer takega pravila je koncept copyleft, ki pri redistribuciji programa ne dopušča dodajanja omejitev, ki bi drugim posameznikom preprečevale uživati osrednje prostosti. To pravilo ne ovira osrednjih svoboščin, temveč jih varuje.

Prostega programja ne gre enačiti z nekomercialnim programjem. Prost program mora biti na voljo tudi za komercialno rabo in distribucijo.

Pravila o tem, v kakšni obliki ali paketu naj se distribuirajo spremenjene različice programa, so sprejemljiva, vse dokler bistveno ne onemogočijo posameznikove prostosti izdajanja spremenjenih različic.

V projektu GNU uporabljajo koncept copyleft, da te prostosti pravno zavarujejo za vsakogar. Ustvarili so ga z namenom preprečiti, da bi se programje med distribucijo olastninilo. »Gre za uporabo pravil intelektualne lastnine za ohranjanje prostega programja in obrambo svobode njegovih uporabnikov. Načela prostega programja dopuščajo, da licence postavljajo določene omejitve glede načina distribuiranja programja. Ideja copylefta je v tem, da se v licenci določi, da nihče, ki distribuira prost program, ne sme nadaljnjim uporabnikom programa omejiti pravice, da ga tudi oni prosto razmnožujejo, spremninjajo in distribuirajo pod enakimi pogoji« [7]. Program se zaščiti s copyleft tako, da se ga najprej avtorsko-pravno zaščiti, nato pa se doda še distribucijske pogoje, ki predstavljajo zakonit instrument, ki daje vsakomur pravico uporabljati, spreminjati in redistribuirati programsko kodo ali katerokoli izpeljano delo. Vendar vse to samo pod pogojem, da ostanejo distribucijski pogoji nespremenjeni. Tako postanejo programska koda in prostosti pravno nerazdružljivi. Vendar obstaja tudi prosto programje, ki ni pod copyleftom.

Včasih lahko prostost distribuiranja izvodov programov po vsem svetu omejijo vladni predpisi posameznih držav na področju kontrole izvoza. Razvijalci programov se tem omejitvam ne morejo izogniti, lahko pa odklonijo, da bi jim bile vsiljene kot pogoji za uporabo njihovega programa. Tako ti predpisi ne bodo prizadeli aktivnosti in posameznikov zunaj jurisdikcije teh držav.

### 2.2.2 Odprta koda

Iniciativa za odprto kodo OSI (Open Source Intitiative)[19] je neprofitna združba, posvečena promociji odprte kode in skrbi za licenciranje odprte kode [1]. »Primarno jo utemeljujejo z bolj pragmatičnimi in manj filozofskimi razlogi« [7]. Odklanjajo stališča, da je lastniško programje a priori nemoralno in poudarjajo predvsem njegove tehnične pomanjkljivosti v primerjavi z odprto kodo [8]. »Na osnovi definicije odprte kode (Open Source Definition, [2]) izvaja združba program certificiranja odprtokodnih licenc in na ta način zagotavlja trdno osnovo za pravilno uporabo pojma odprta koda, vzpodbuja širjenje uporabe odprtokodnih licenc in s tem pospešuje razvoj odprtokodnih projektov. Na ta način trdno podpira osnovno idejo odprte kode, ki vidi v dovoljenju za prosto branje, redistribucijo in spreminjanje programske opreme neslutene razvojne možnosti. Prepričani so, da je prišel čas za preboj odprte kode na komercialno področje. V najširšem smislu je odprta koda vsaka programska oprema, za katero je omogočen dostop do programske kode. V smislu OSI definicije odprte kode pa to še zdaleč ne zadošča. Z oznako OSI odprta koda se lahko ponaša le programska oprema, pri kateri pravila distribucije oziroma licenciranja izpolnjujejo 10 kriterijev, zapisanih v definiciji odprte kode« [32]:

1. **Svobodna distribucija**

   Licenca mora dovoljevati prosto prodajo ali predajo programske opreme kot komponente združenih programskih paketov in ne sme zahtevati nikakršnih plačil.

2. **Izvorna koda**

   Program mora vsebovati izvorno kodo. Če se program ne distribuira skupaj z izvorno kodo, pa mora biti ta brezplačno javno dostopna. Oblika izvorne kode mora omogočati spreminjanje. Kakršnokoli zakrivanje ali drugačno oteževanje njene uporabe je prepovedano. Vmesne oblike, kot je izhod (output) prevajanika, niso dovoljene.

3. **Izpeljana dela**

   Licenca mora dovoljevati spreminjanje in izdelavo izpeljanih programskih rešitev, ki se distribuirajo pod enakimi licenčnimi pogoji kot to velja za originalno programsko opremo.

4. **Integriteta avtorja izvorne kode**

   Licenca lahko zahteva, da morajo biti izpeljani izdelki distribuirani pod spremenjenim nazivom ali spremenjeno oznako verzije. Prepoved distribucije izvorne kode v spremenjeni obliki je sprejemljiva le v primeru, da licenca dovoljuje distribucijo

[19] spletna stran OSI: http://www.opensource.org

popravkov z namenom spreminjanja programa v času prevajanja. Nedvoumno mora biti dovoljena distribucija programske izvršne kode.

5. **Prepoved diskriminacije posameznikov in skupin**

   Licenca ne sme diskriminirati nobenega posameznika ali skupine posameznikov.

6. **Prepoved diskriminacije posameznih področij dejavnosti**

   Licenca ne sme nikogar omejevati pri uporabi programa na posameznem prodročju dejavnosti. Na primer, ne sme omejevati uporabe programa za poslovne namene, komercialne rabe programja itd.

7. **Distribucija licence**

   Licenčne pravice se morajo nanašati na vsakogar, ki prepiše ali prejme programsko rešitev, brez dodatnih postopkov licenciranja.

8. **Licenca ne sme biti specifična za produkt**

   Pravice, vezane na program, ne smejo biti odvisne od tega, ali je program del določene programske distribucije. Če je program odstranjen iz distribucije in uporabljen ali distribuiran naprej pod pogoji licence programa, morajo vse stranke, katerim je bil program distribuiran, imeti enake pravice kot tiste, ki so jim bile dodeljene pravice za originalno programsko distribucijo.

9. **Licenca ne sme omejevati druge programske opreme**

   Licenca ne sme postavljati omejitev nad programsko opremo, ki je distribuirana skupaj z licencirano programsko opremo. Na primer, licenca ne sme zahtevati, da mora biti vsa programska oprema, ki je distribuirana na istem mediju prav tako odprtokodna programska oprema.

10. **Licenca mora biti nevtralna do tehnologije**

    Prevzemanje licenc ne sme biti omejeno na posamezne tehnološke rešitve ali vmesnike.

Danes sta gibanje za prosto programje in gibanje za odprto kodo ločeni. Imata različne poglede in cilje, čeprav sodelujeta skupaj na nekaterih projektih. Najpomembnejša razlika med njima je v njunih vrednotah, načinu gledanja na svet ter v licenčnih zahtevah. Gibanje za odprto kodo dovoljuje več svobode pri licencah; zanj je vprašanje, ali naj bo programska oprema odprtokodna, praktično in ne etično vprašanje. Na drugi strani gibanje za prosto programje označuje neprosto programje kot neoptimalno rešitev. Zanje je neprosto programje družbeni problem in prosto programje rešitev. V nadaljevanju diplomske naloge bom uporabljal izraza odprta koda in prosto programje kot isti pojem, v smislu programske opreme, ki jo je možno svobodno uporabljati, spreminjati in razširjati naprej.

## *2.3 Licenčni modeli prostega programja*

V zadnjih nekaj letih je nastala množica licenc odprte kode. Večina novejših licenc je spremenjenih tako, da služijo določenemu poslovnemu modelu. Licence podjetij kot so IBM[20], Sun[21] in Netscape Corporation sodijo v to kategorijo. Vedno težje je obdržati pregled nad različnimi licencami in njihovimi komaj opaznimi razlikami. Tako GNU projekt kot Iniciativa za odprto kodo (OSI) sta skupili širok seznam licenc, ki ustrezajo kriterijem odprte kode[22]. Ta seznam se stalno povečuje. Nekatere od bolj znanih licenc so:

- GNU General Public Licence (GPL)
- GNU Library ali »Lesser« Public Licence (LGPL)
- BSD Licence
- MIT Licence
- Mozilla Public Licence (MPL)
- Q Public Licence (QPL)
- Artistic Licence
- IBM Public Licence

### 2.3.1 GNU Public Licence – GPL[23]

Večina licenc programske opreme prepoveduje distribucijo in spreminjanje programja. Nasprotno GPL zagotavlja svobodo distribuiranja in spreminjanja prostega programja - poskrbi za to, da je programje prosto za vse njegove uporabnike [3]. GPL je najbolj pomembna licenca prostega programja, saj je večina prostega programja distribuirana pod njenimi pogoji. Glavni razlog njene popularnosti je njen virusni efekt, to je koncept copyleft.

| | **GPL (Verzija 2)** |
|---|---|
| **Uporaba licence** | Licenca se lahko uporabi za kakršnokoli programsko opremo, vendar se je ne sme modificirati |
| **Plačila (ang. fees)** | Licenca lahko zahteva plačilo za fizično dejanje oskrbe stranke s kopijo in lahko zaračuna plačilo za vsako dodatno garancijsko zaščito, ki jo ponuja. |
| **Distribucija** | Redistribucija (vključno z vsemi spremembami) mora biti v skladu s pogoji GPL. |
| **Razpoložljivost izvorne kode** | Vsaka distribucija mora vključevati izvorno kodo, ali pa mora licenca ponuditi oskrbo izvorne kode na zahtevo stranke za obdobje vsaj treh let. |
| **Uporaba z drugo programsko opemo** | Vsako delo, ki v celoti ali deloma vsebuje, ali izhaja iz licenciranega programa, mora biti distribuirano pod GPL licenco. To ne velja za neodvisna in ločena dela, ki ne izhajajo iz |

[20] http://www.ibm.com
[21] htpp://java.sun.com
[22] število licenc, ki jih OSI odobrava¸ je do danes doseglo številko 58. (http://www.opensource.org/licenses/)
[23] http://www.opensource.org/licenses/gpl-license.php

| | |
|---|---|
| | licenciranega programa. |
| **Garancija in odgovornost** | Nobene garancije in popolna izključenost odgovornosti v skladu z zakonskimi okviri. Licenca lahko ponuja garancijo svojim strankam, če se tako odloči. |
| **Prenehanje** | Licenca avtomatsko preneha veljati ob prekršitvi njenih pogojev s strani imetnika licence. |

**Tabela 2.1 GPL licenca (Vir [13])**

### 2.3.2 Lesser GPL - LGPL[24]

Ta licenca se razlikuje od GPL v enem pomembnem vidiku: dovoljuje povezovanje z neprostimi moduli. Prvotno je bila oblikovana za standardne knjižnice, z namenom pospeševanja prisvajanja prostega programja, saj tako licencirane knjižnice nudijo priložnost lastniškim programom, da se izvajajo v sistemu prostega programja. FSF je mnenja, da omogočanje knjižnic pod LGPL škoduje namenu in načelom prostega programja, zato je ne podpirajo.

| | **LGPL (Verzija 2.1)** |
|---|---|
| **Uporaba licence** | Licenca se lahko uporabi za kakršnokoli programsko opremo, vendar je primarno namenjena programskim knjižnicam. Licence ni dovoljeno modificirati. |
| **Plačila (ang. fees)** | Licenca lahko zahteva plačilo za fizično dejanje oskrbe stranke s kopijo in lahko zaračuna plačilo za vsako dodatno garancijsko zaščito, ki jo ponuja. |
| **Distribucija** | Redistribucija (vključno z vsemi spremembami) mora biti v skladu s pogoji LGPL ali GPL. |
| **Razpoložljivost izvorne kode** | Vsaka distribucija mora vključevati izvorno kodo, ali pa mora biti izvorna koda dostopna na istem mestu kot objektna koda. |
| **Uporaba z drugo programsko opemo** | Delo, ki uporablja knjižnico tako, da je prevedeno (compiled) ali povezano (linked) z njo in ne vsebuje izpeljav knjižnice, je izvzeto iz licence. Vendar, če program vsebuje dele knjižnice kot rezultat takega prevajanja ali povezovanja, se lahko distribuira pod katerimikoli licenčnimi pogoji, pod pogojem, da dovoljujejo modifikacije in obratni inženiring[25] za potrebe stranke. |
| **Garancija in odgovornost** | Nobene garancije in popolna izključenost odgovornosti v skladu z zakonskimi okviri. Licenca lahko ponuja garancijo svojim strankam, če se tako odloči. |
| **Prenehanje** | Licenca avtomatsko preneha veljati ob prekršitvi njenih pogojev s strani imetnika licence. |

**Tabela 2.2 LGPL licenca (Vir [13])**

### 2.3.3 BSD licenca (Berkley Software Distribution)[26]

Ta licenca dovoljuje, da se s programsko opremo naredi skoraj vse. Najbolj pomembno dovoljenje, ki ga v GPL ni, je da se lahko BSD-licencirane spremembe naredijo zasebne. Drugače povedano, lahko se spreminja pridobljeno izvorno kodo BSD-licenciranega

[24] http://opensource.org/licenses/lgpl-license.php
[25] pri obratnem inženiringu gre za raziskovanje določenega dela programja, da bi videli kako deluje in posnemali dobre rešitve in ideje
[26] http://opensource.org/licenses/bsd-license.php

programa in se ga potem prodaja v binarni obliki, ne da bi se poleg distribuirala tudi spremenjena izvorna koda. To je še vedno odprta koda, saj definicija odprte kode ne zahteva, da imajo izpeljana dela originalno licenco.

| | **BSD (Julij 1999)** |
|---|---|
| **Uporaba licence** | Licenca se lahko uporabi za kakršnokoli programsko opremo. |
| **Plačila (ang. fees)** | Nobenih posebnih predpisov. |
| **Distribucija** | Redistribucija je možna pod pogojem, da so pogoji BSD licence spoštovani. |
| **Razpoložljivost izvorne kode** | Nobenih zahtev glede razpoložljivosti izvorne kode. |
| **Uporaba z drugo programsko opemo** | Nobenih posebnih predpisov. |
| **Garancija in odgovornost** | Nobene garancije in popolna izključenost odgovornosti. |
| **Prenehanje** | Nobenih posebnih predpisov. |

**Tabela 2.3 BSD licenca (Vir [13])**

### 2.3.4 Mozilla javna licenca (Mozilla Public license - MPL)[27]

MPL licenco je razvila združba Netscape Corporation, ko je objavila svoj brskalnik Netscape Navigator v odprti kodi. Licenca omogoča, da postanejo določene modifikacije programa zasebne. Definicija modifikacije programa v MPL je, da je to vsaka sprememba v datoteki, ki je del originalnega programa, in vsaka nova datoteka, v katero je bil prepisan določen del izvorne kode originalnega programa. Iz tega sledi, da lahko ostanejo zasebne samo datoteke, ki v celoti vsebujejo povsem novo izvorno kodo (ni bila vzeta iz originalnega programa). Veliko podjetij je sprejelo variacijo MPL za svoje programe. Tako so nastale licence: Netscape Public Licence, Interbase Licence, Nokia Open Source Licence itd.

| | **MPL (Verzija 1.1)** |
|---|---|
| **Uporaba licence** | Licenca se lahko modificira in uporabi za drugo programsko opremo, pod pogojem, da se preimenuje in da so v novi licenci jasno razvidni drugačni pogoji. |
| **Plačila (ang. fees)** | Licenca lahko zaračuna plačilo za vsako dodatno garancijo, podporo, zavarovanje, ki ga ponuja. |
| **Distribucija** | Modifikacije se morajo licencirati pod MPL. Zajeta koda (originalnega licenciranega programa in modifikacije), ki ni izdana kot izvorna koda, se lahko distribuira pod drugimi licenčnimi pogoji, ki pa morajo biti skladni z MPL. |
| **Razpoložljivost izvorne kode** | Vse modifikacije morajo biti dostopne v izvorni kodi, bodisi na istem mediju kot ali ločeno. Za slednje velja, da morajo modifikacije ostati dostopne vsaj 12 mesecev, ali vsaj 6 mesecev po tem, ko je bila izdana naslednja verzija. |
| **Uporaba z drugo programsko opemo** | Licenca lahko združuje programsko opremo z drugimi programi, pri čemer le-ti niso pod vplivom MPL. Vendar mora biti licenca še vedno skladna z MPL za vso kodo, ki jo pokriva. |
| **Garancija in odgovornost** | Nobene garancije in popolna izključenost odgovornosti, razen odgovornosti za smrt ali poškodbe kot posledica malomarnosti v skladu z zakonskimi okviri. |

[27] http://opensource.org/licenses/mozilla1.1.php

| | |
|---|---|
| **Prenehanje** | Licenca avtomatsko preneha veljati, če se v tridesetih dneh po ugotovitvi kršitve njenih pogojev le-te ne odpravijo. |

**Tabela 2.4 MPL licenca (Vir [13])**

## *2.4 Odprtokodne skupnosti*

Odprtokodne skupnosti so danes ene najbolj uspešnih in hkrati najslabše razumljenih primerov visoko kakovostnega in učinkovitega sodelovanja na medmrežju. Druge skupnosti bi lahko ogromno pridobile z razumevanjem delovanja odprtokodnih skupnosti.

Prosto programje se razvija v rahlo organiziranih, »ad-hoc« skupnostih, ki jih tvorijo posamezniki iz celega sveta. Le-ti se med seboj osebno ne poznajo, vendar jih združuje zavezanost k istemu cilju. Taki mešanici posameznikov učinkovito uspeva opravljati tako težko nalogo, kot je izgradnja visoko kakovostne programske opreme. Uspeh odprtega programja je prisilil ljudi v ponoven razmislek o tradicionalnem pogledu na razvoj programja, individualne psihologije in organizacijske dinamike [10].

### 2.4.1 Trg

Na medmerežju trenutno obstajajo trije večji repozitoriji prostega programja:

- GNU Free Software Directory[28],
- SourceForge[29],
- Freshmeat[30].

Prvega vzdržuje organizacija FSF (Free Software Foundation) in vsebuje samo projekte, ki so razpoložjivi pod GPL licenco. Druga dva repozitorija ima v lasti Open Source Development Network (OSDN)[31].

| **Repozitorij** | **Število projektov** |
|---|---|
| SourceForge | 88,977 |
| Freshmeat | 34,877 |
| GNU Free Software Directory | 3,567 |

**Tabela 2.5 Število projektov po repozitorijih (stanje Oktober 2004)**

[28] http://www.gnu.org/directory
[29] http://sourceforge.net
[30] http://freshmeat.net
[31] http://www.osdn.com

Med naštetimi repozitoriji Freshmeat pokriva najširši spekter prostega programja in vključuje tudi nekatere programske pakete, ki se nahajajo tako na SourceForge kot na GNU Free Software directory [10].

### 2.4.2 Demografija

V zadnjih treh letih so številne študije skušale odgovoriti na vprašanje, kdo razvija prosto programje in zakaj. Tri še posebej izstopajo:

- Who is doing it (WIDI) študija (2001), ki jo je izvedla skupina raziskovalcev na Tehnični fakulteti v Berlinu
- The Boston Consulting Group/OSDN (BCG/OSDN) »Heker« študija (2002)
- Free/Libre and Open Source Software študija (FLOSS) (2002), ki jo je izvedel International Institute of Infonomics na Maastrichtski univerzi (Nizozemska)

Rezultati vseh treh so zelo podobni in prikazujejo jasno sliko tipičnega razvijalca prostega programja[32]:

- **Pretežno moški.** Vse tri študije podajajo, da je bilo več kot 98 odstotkov anketiranih moških. (FLOSS 8, BCG 21, WIDI Part 1).
- **Pretežno Generacija X.** Več kot 70 odstotkov anketiranih v vseh treh študijah je bilo starih med 22 in 37 leti. Povprečna starost je bila v območju od 27 do 30 let. (FLOSS 9, BCG 21, WIDI Part 1)
- **Koncentrirani v ZDA in Evropi.** Več kot 80 odstotkov anketiranih v vseh študijah prihaja iz ali živi v ZDA in Evropi. (FLOSS 16-17, BCG 21, WIDI Part 1)
- **IT strokovnjaki.** Več kot 50 odstotkov anketiranih v vseh študijah dela v IT[33]. Študentje so na drugem mestu z 20 do 30 odstotki. (FLOSS 13, BCG 25, WIDI Part 3)
- **Večinoma z univerzitetno in srednješolsko izobrazbo.** WIDI in FLOSS navajata, da ima od 33 do 46 odstotkov anketiranih univerzitetno izobrazbo, medtem ko ima 17 do 24 odstotkov srednješolsko izobrazbo. FLOSS poroča, da ima 28 odstotkov anketiranih magistrsko izobrazbo, medtem ko jih je pri WIDI poročilu samo 12 odstotkov. (FLOSS 12, WIDI Part 3)
- **Nepopolna udeležba.** Od 34 do 48 odstotkov anketiranih porabi manj kot 5 ur na teden za delo na projektih odprte kode. 9 do 15 odstotkov je takih, ki porabijo 20 do

[32] povzeto po [10]
[33] informacijska tehnologija

40 ur na teden in samo 5 do 7 odstotkov anketiranih dela več kot 40 ur na teden na projektih odprte kode. (FLOSS 21, BCG 23, WIDI Part 4)

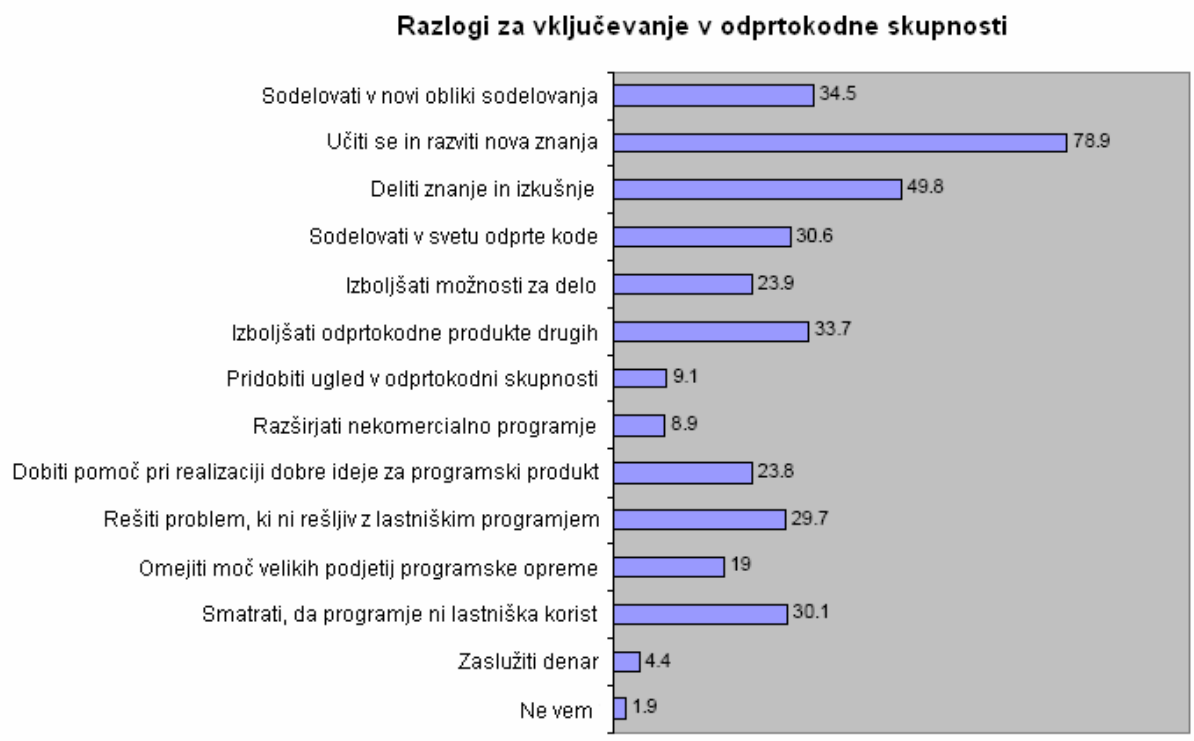
### 2.4.3 Motivacija

Literatura o človeški motivaciji razlikuje med notranjo in zunanjo motivacjio. Notranja motivacija je motivacija, ki izvira iz zanimanja za neko aktivnost. To pomeni, da se notranje motiviran človek ne oklepa zunanjih ciljev, kot so nagrada, dobra ocena, priznanje in uveljavitev v javnosti, saj so zanj značilne notranje motivacijske spodbude (radovednost, interesi, vzburjenje, zanos, pozitivna samopodoba), ki neposredno spodbudijo motivacijski proces [17]. Kadar je posameznik notranje motiviran, ne potrebuje spodbud ali kaznovanja, ker je »že aktivnost sama po sebi nagrada« [18]. Za zunanjo motivacijo so značilne zunanje motivacijske spodbude, ki izhajajo iz okolja, so posredne, uporablja jih nekdo od zunaj (starši, učitelji, sošolci, vrstniki), da bi z njimi sprožil motivacijski proces [17]. Zunanje motiviran človek deluje zaradi zunanjih posledic (pohvala, graja, nagrada, kazen, preverjanje in ocenjevanje), sama aktivnost ga ne zanima, delo je le »sredstvo za doseganje pozitivnih in izogibanje negativnih posledic« [19]. Posameznika motivira nek pričakovan rezultat, ki si ga postavi za cilj delovanja, le-ta je pomembnejši od procesa, vir podkrepitve pa prihaja od zunaj. Če vir zunanje podkrepitve izgine, dejavnost preneha [20].

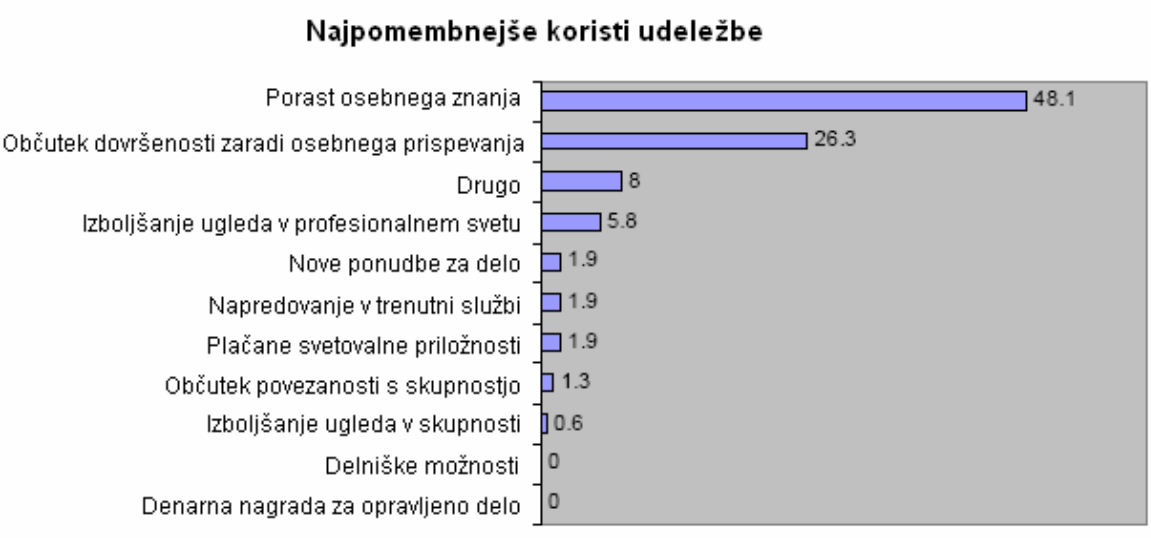
Rezultati FLOSS in BCG/OSDN študij o motivaciji programerjev v odprtokodnih skupnostih, ki jih bom predstavil v nadaljevanju, prikazujejo, da pri programerjih prevladuje notranja motivacija.

FLOSS študija je ugotovila, da 46 odstotkov vseh anketiranih ne dobi nobenega plačila (posrednega ali neposrednega) za svoje delo. 16 odstotkov je plačanih za razvoj odprtokodne programske opreme, 18 odstotkov je plačanih za administracijo, 12 odstotkov pa je plačanih za podporo (uporabniško, tehnično). 26 odstotkov anketiranih je zatrdilo, da niso prejeli nobene posredne finančne kompenzacije za svoje delo na odprti kodi in 18 odstotkov je dejalo, da jim je delo na odprti kodi pripomoglo k pridobitvi zaposlitve. Tako FLOSS kot BCG/OSDN študije so ugotovile, da je poglaviten razlog posameznikov za vključevanje in nadaljevanje dela v odprtokodnih projektih, razširiti in deliti svoje znanje. Tako je 93 odstotkov anketiranih BCG/OSDN študije dejalo, da je širjenje osebnega znanja korist sodelovanja in za 48 odstotkov anketiranih je bila to celo najpomembnejša korist. Prav tako je

79 odstotkov anketiranih FLOSS študije dejalo, da so se pridružili z namenom učenja in razvijanja novih spretnosti, 50 odstotkov pa je dejalo, da so se pridružili z namenom deljenja svojih znanj in spretnosti. (FLOSS 45, BCG 17). Sliki 2.2 in 2.3 prikazujeta rezultat FLOSS in BCG/OSDN študij.



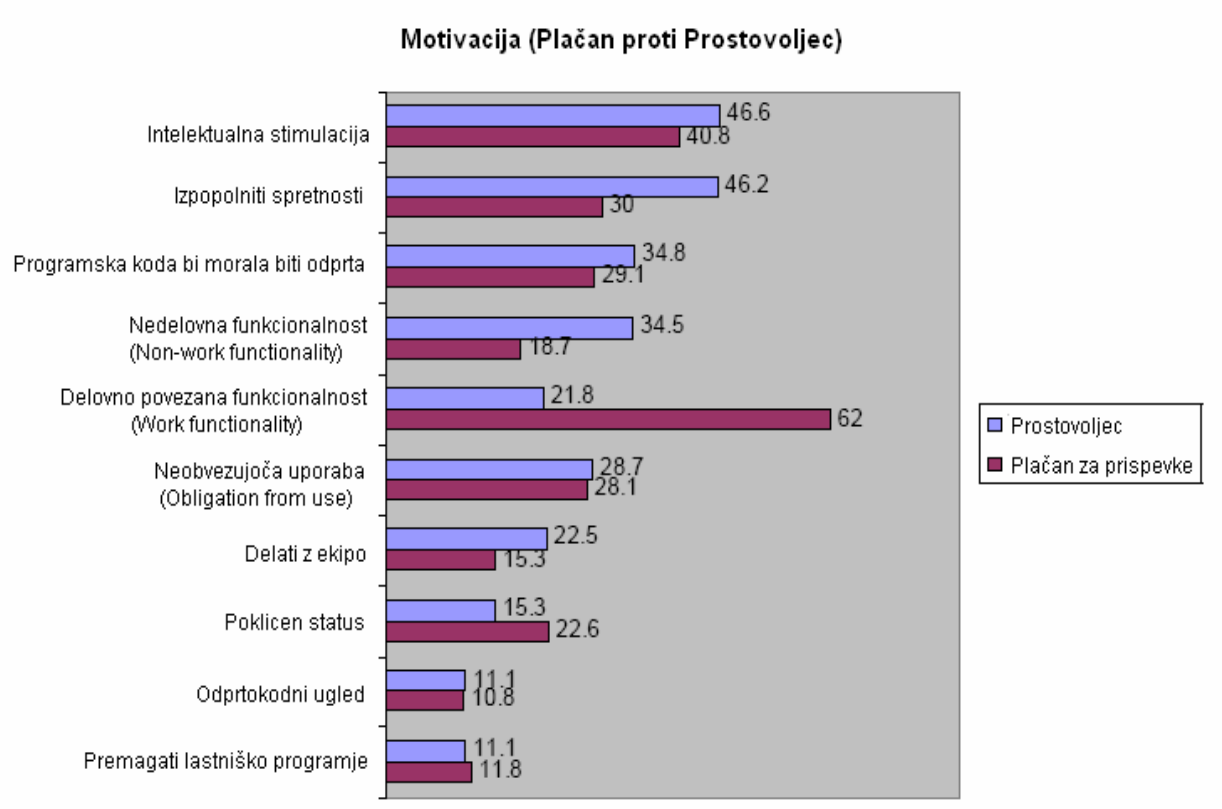

**Slika 2.2 Razlogi za vključevanje v odprtokodne skupnosti**




**Slika 2.3 Najpomembnejše koristi sodelovanja v odprtokodni skupnosti**

BCG/OSDN študija je primerjala različne motivacije prostovoljnih darovalcev odprte kode in tistih, ki so plačani za delo na odprtokodnih projektih. Kot je razvidno iz slike 2.4 so

motivacije zelo primerljive, razen ene – »delovno povezana funkcionalnost«. Tisti, ki so plačani za delo na odprtokodnem projektu, imajo precej večjo motivacijo opraviti svoje delo dobro in učinkovito.




**Slika 2.4 Razlika med motivacijo prostovoljcev in tistih, ki so plačani za odprto kodo**

Programerji sodelujejo v odprtokodnih skupnostih z namenom pridobiti ugled med svojimi vrstniki, signalizirati svoj talent podjetjem in proučevati kodo drugih razvijalcev in se s tem kaj novega naučiti. Te cilje lahko dosežejo le, če je koda, ki jo prispevajo odprta in razpoložljiva za vse.

### 2.4.4 Upravljanje z znanjem v odprtokodnih skupnostih[34]

Za delovanje, rast in razmišljanje »možganov« skupnosti so ključni določeni procesi, ki so predstavljeni v nadaljevanju.

[34] povzeto po [21]

***Omogočanje ponovnih izkušenj z zmanjševanjem kompleksnosti in »porazdeljen« spomin skupine (transactive group memory[35])***

Grajenje skladišča informacij (spomina) in organiziranje novih vsebin je temelj sistema znanja skupnosti. Za lažjo prebavo ogromnih količin znanja, so znotraj skupnosti implementirane ustrezne tehnologije in značilnosti delovnih aktivnosti (ang. task-related features), ki zmanjšujejo kompleksnost. Kot primer lahko navedem uporabo modularne strukture aktivnosti, hranjenje zgodovine programske kode v CVS[36] repozitoriju in prenos znanja s posameznikov v porazdeljen spomin skupine, kjer člani skupnosti vedo, kje najti določeno informacijo.

Za lažje dojemanje razvijalci dodajajo komentarje svoji izvorni kodi, kar sproža procese ponovnega razmišljanja in ponovnih izkušenj pri drugih članih skupnosti. Najpomembnejši steber sistema znanja skupnosti predstavljajo poštni seznami[37] (ang. mailing lists). To so platforme, kjer se odvijajo vse pomembnejše razprave in diskusije. Arhivirajo se kot porazdeljen spomin skupine učeče skupnosti.

***Omogočanje ponovnih izkušenj z vodenjem, odprtostjo in legitimnim obrobnim sodelovanjem***

Novi člani so v skupnost integrirani preko standardiziranega postopka in strogega vodstva pri prevzemanju nalog in kulturnih norm. Novince se spodbuja k opazovanju običajne prakse in komunikacij, s čimer se pri njih poskuša vzgojiti procese ponovnega razmišljanja in ponovnih izkušenj. Zaradi reflektivne narave komentirane izvorne kode, interaktivnih učbenikov (ang. tutorials) ter družbene interakcije preko asinhronih komunikacijskih orodij[38], je učenje možno tudi brez medosebne interakcije. Odprtost do vseh orodij in komunikacij je ključ za deljenje znanja.

***Omogočanje ponovnih izkušenj z asinhronimi komunikacijami in virtualnim eksperimentiranjem***

Asinhrona komunikacijska orodja se primarno uporabljajo za pridobivanje novega znanja. Sinhrona komunikacijska orodja kot so avdio, video konference, klepetanje (ang. chat),

[35] porazdeljen spomin skupine je skupinski spominski sistem, ki podrobno razlaga strokovno znanje posameznikov v skupini skupaj z zavestjo kdo kaj ve v skupini

[36] CVS – Concurrent Version System: Sistem za upravljanje z verzijami sestavnih delov programa

[37] zbirka imen in naslovov, ki jih uporabi posameznik ali organizacija za pošiljanje gradiva množici prejemnikov

[38] to so komunikacijska orodja, ki omogočajo komunikacijo in sodelovanje v načinu »različen čas-različen prostor« skozi daljše časovno obdobje. Ne gre za komunikacijo v realnem času. Primeri takih komunikacijskih orodij so spletni dnevniki (spletne strani izbrane vsebine, ki jih uporabniki pogosto ažurirajo), diskusijski forumi, sporočanje preko elektronske pošte itd.

takojšnje sporočanje (ang. instant messaging) se uporabljajo samo za koordinacijo aktivnosti in razpravljanje o rešitvah. Uporaba sinhronih orodij v druge namene se izogiba, saj ovirajo nadaljno premišljevanje in proučevanje v obliki dvojne zanke in s tem pridobivanje novega znanja. Razvijalci z uporabo zgodbic in praktičnih scenarijev oblikujejo virtualni svet, ki predstavlja bodočo realizacijo njihovih idej. To konkretiziranje idej daje otipljivost, ki jo skupina posameznikov potrebuje za sooblikovanje skupnega razumevanja in predstave njihovih bodočih akcij.

Funkcije upravljanja z znanjem v tehnoloških skupnostih, med katere spadajo tudi odprtokodne skupnosti, temeljijo na pripovedovanju, družbeni konstrukciji in sodelovanju [22]. Prek tega si člani skupnosti pomagajo pri dojemanju nestandardnih informacij ter gradijo svoj svet s svojimi simboli, jezikom in družbenim redom. Z opiranjem na skupnost, njeni člani prihranijo pri iskanju in eksperimentiranju. V tehnoloških skupnostih člani sodelujejo s pošiljanjem in odgovarjanjem na sporočila, organiziranjem razprav in ponujanjem raznih interesnih aktivnosti ostalim članom. Za tehnološke skupnosti so značilne določene karakteristike in vedenjske norme, predstavljene v nadaljevanju:

- Delovne navdušenosti v skupnosti je v izobilju; člani so pripravljeni žrtvovati svoj čas (ali denar) in prispevati k namenu skupnosti, organizirati skupinske sestanke in širiti namen, idejo skupnosti. Uporaba tehnologije ni namenjena dobičkonosnosti ali kariernemu napredovanju posameznikov, temveč gre za njihov hobij in strast. Zvesti člani ne tolerirajo kakršnihkoli opazk ali kritik glede tehnologije in so se jim pripravljeni zoperstaviti; njihove reakcije so lahko logične ali pa čustvene.
- Člani skupnosti z več znanja, spretnosti in izkušenj so posebej spoštovani. Mnenja strokovnjakov imajo posebno težo in pomen. Zato v interaktivnih forumih vedno obstajajo določeni posamezniki, katerih nasvet ali mnenje je vredno več.
- Filozofija »Preberi odličen priročnik« (RTFM-Read the Fine Manual): Člani skupnosti verjamejo, da bi moral vsak nov uporabnik prebrati dokumente kot so Pogosto zastavljena vprašanja (ang. FAQ), priročniki (ang. HowTo, Manual) in se iz njih čim več naučiti, saj je po njihovem mnenju to najboljši način za pridobitev ustreznega znanja. S tem razlogom se vsakega novinca, ki postavi neko osnovno vprašanje usmeri k ustreznemu dokumentu, kjer lahko najde odgovor na svoje vprašanje. Samo v primeru, če je problem kompleksnejši ali zanj ni moč najti odgovora v dokumentaciji, bo na pomoč priskočil ustrezen strokovnjak in predlagal

rešitev. Strokovnjaki neradi rešujejo trivialne probleme, zato se je za pomoč pogosto bolje obrniti na uporabnike s srednjim znanjem, ki so bolj pripravljeni pomagati.

- Tehnični nasveti novim uporabnokom niso dani na teoretnični osnovi, temveč s stališča praktičnega znanja in izkušenj. Skupnost ne spoštuje podajanja zavajajočih ali napačnih nasvetov novincem in takšna dejanja obsodi z opozorilom ali kaznijo (npr. izključitev iz klepetalnice).[39]

## *2.5 Odprtokodni projekti*

Odprtokodno skupnost je zelo težko raziskovati kot abstrakten socialen fenomen. Težko je razločiti, kdo je in kdo ni del nje. K sreči lahko odprtokodne projekte opazujemo in analiziramo zaradi njihove prisotnosti na medmrežju in njihovih javno dostopnih komunikacij. Kaj torej je odprtokodni projekt? To je vsaka skupina ljudi (ali izključni posamezniki), ki razvija programsko opremo in posreduje svoje rezultate javnosti pod odprtokodno licenco.

### 2.5.1 Razvojni model (Cathedral and Bazaar)

Eric Raymond je v svojem eseju z naslovom *The Cathedral and the Bazaar*[40] predstavil razvojni model odprtokodnih projektov, ki je nastal na podlagi njegovih opažanj procesa razvoja Linux jedra in njegovih izkušenj pri upravljanju odprtokodnih projektov. Med seboj je primerjal dva modela razvoja programske opreme, ki jih je poimenoval Cathedral in Bazaar.

Model, tipiziran s Cathedral, predstavlja razvoj programske opreme na podlagi »a-priori« projekta, ki predpisuje vse funkcije in značilnosti, ki bodo vključene v končnem produktu[11]. Delo programerjev je centralno koordinirano in nadzorovano s strani vodilnega načrtovalca ali ekipe načrtovalcev. Nobeno programje ni izdano predčasno. Izvorna koda je zelo lastniška in se ne izdaja. Razvojni preizkuševalci (ang. beta testers)[41] so zelo pazljivo izbrani, zavezani k molčečnosti in zaupnosti. Razvojno preizkušanje je namenjeno samo iskanju hroščev[24]. Gre za centraliziran, korporacijski model razvoja programske opreme, ki dominira v industriji.

Nasprotno se pri Bazaar modelu programska oprema razvije iz nestukturiranega evolucijskega procesa[11]. Vsi delajo vse, ves čas. Izvorna koda je izdana skupnosti. Delo koordinira osrednja skupina ljudi, ki ima vedno zadnjo besedo pri odločanju. Program na nek način ves čas ostaja v razvojni (beta) fazi, verzije se zelo pogosto izdajajo, včasih tudi po večkrat na

[39] povzeto po [23]
[40] celoten esej je dostopen na naslovu: http://www.linuxresources.com/Eric/cathedral.html
[41] posamezniki, ki so zadolženi za testiranje razvojne (nedokončane) verzije programa.

dan. Skupnost uporabnikov ni namenjena samo podajanju predlogov za izboljšave in poročanju o hroščih, temveč naj bi hrošče pomagali odstranjevati in razširjati programsko kodo z dodajanjem izboljšav in nadgradenj [24]. Bazaar predstavlja decentraliziran model. Glavne prednosti, ki jih po mnenju E. Raymonda nudi odprtokodni (Bazaar) razvoj programske opreme, so [25]:

- Brook-ov zakon za ta način razvoja ne velja; Brook-ov zakon pravi, da se storilnost programerskega dela na projektu linearno (proporcionalno) povečuje s številom programerjev (N), kompleksnost projekta pa se zaradi komunikacijskih in organizacijskih zahtev kvadratično povečuje glede na število programerjev ($N^2$). V odprtokodnih projektih je učinkovita komunikacijska infrastruktura vzpostavljena že od samega začetka, zato so posledice Brook-ovega zakona pri odprtokodnem razvoju manj kritične.
- Velja Linus-ov[42] zakon, ki pravi »več kot je ljudi, ki imajo vpogled v izvorno kodo, večja je verjetnost, da se odkrije napaka« (ang. *Given enough eyeballs, all bugs are shallow*). Pri običajnem razvoju programske opreme je za testiranje zadolžena le peščica uporabnikov. Pri odprtokodnem razvoju pa produkt poleg celotne razvojne ekipe testirajo tudi vsi končni uporabniki. S tem je produkt preizkušen na različnih strojnih in programskih okoljih.
- Frekvenca izdajanja verzij; Odprtokodni projekti nimajo časovnega pritiska za izdaje verzij programske opreme, kar ima svoje prednosti in slabosti. Prednost pogostih izdaj je hitro odkrivanje napak v funkcionalnostih in arhitekturi ter njihovo odstranjevanje z izdajo novih različic. Slabost je, da to pri uporabnikih pogosto povzroča zmedo, saj kmalu izgubijo pregled nad vsemi izdanimi verzijami in novostmi ter spremembami v njih.

### 2.5.2 Uporabniki in njihove vloge

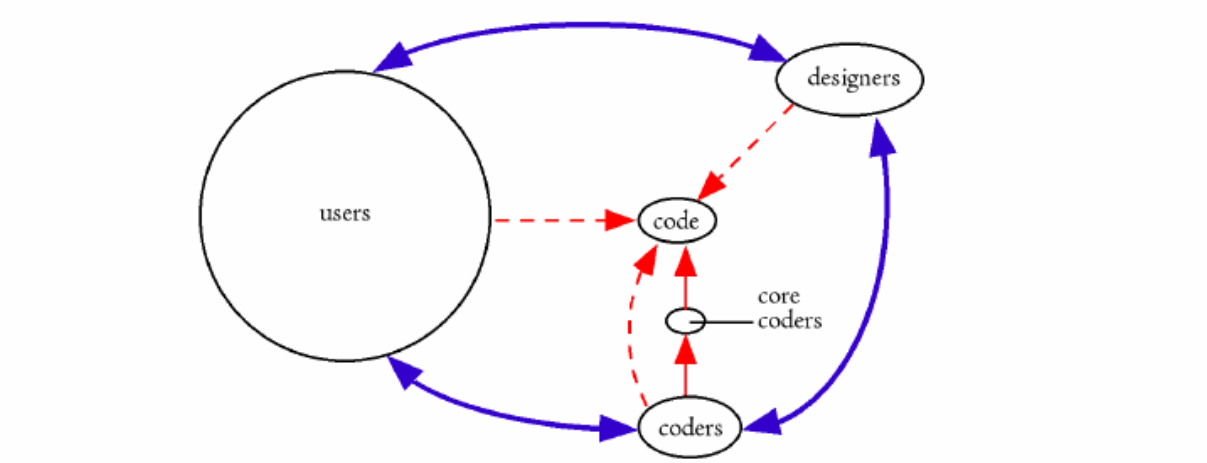
Uporabniki prostega programja prihajajo iz zelo raznolikih družbenih okolij, kot so:

- izobraževalne institucije,
- raziskovalne institucije,
- distributerji programske opreme,
- komercialne organizacije,

[42] ime je dobil po Linusu Torvaldsu

- zasebni uporabniki in
- vladne institucije.

Posamezniki, vključeni v odprtokodni projekt, prevzemajo različne vloge. Skozi čas lahko nastopajo kot navadni uporabniki, načrtovalci ali programerji. Diagram na sliki 2.5 prikazuje tipične uporabnike in njihove medsebojne relacije ter njihovo povezavo s programsko kodo.




**Slika 2.5 Programska koda v žarišču**

Slika 2.5 prikazuje programsko kodo, ki je v središču, vse se odvija okrog nje; uporabniki, načrtovalci in programerji lahko kodo pregledujejo, vendar samo tako imenovani osrednji, glavni programerji (ang. core coder) lahko z njo direktno manipulirajo. Programerji s kodo manipulirajo bodisi s pisanjem nove kode (z namenom implementirati neko novo načrtovano funkcijo) bodisi s preučevanjem in popravljanjem obstoječe kode, ki jo je napisal nekdo drug. Tanka, polna črta nakazuje izvajanje sprememb v izvorni kodi, prekinjena črta nakazuje pregledovanje izvorne kode in interakcijo s programom, debelejša polna črta pa nakazuje komunikacije med ljudmi v posameznih vlogah [26].

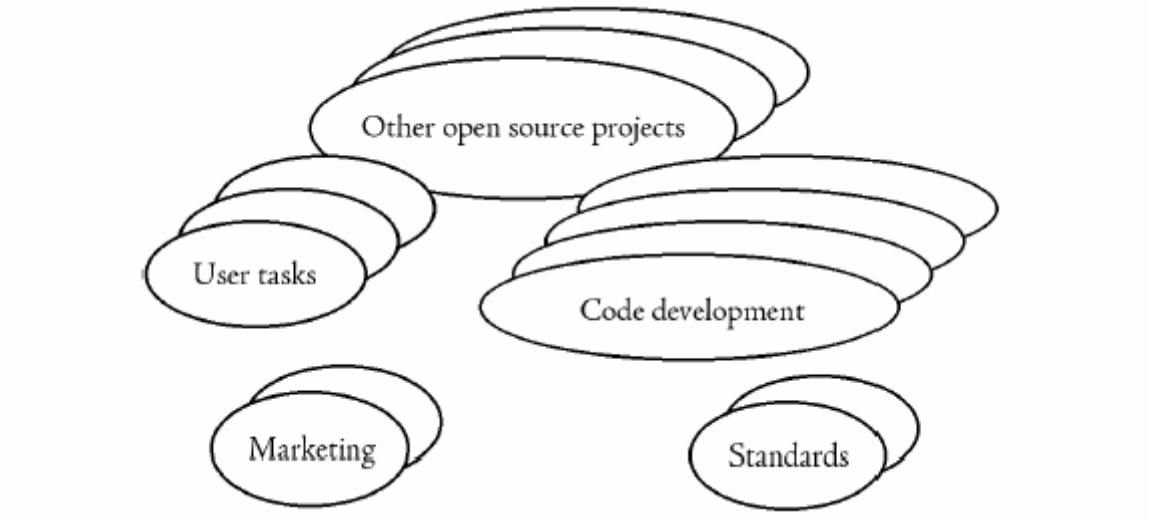
Pripadniki skupnosti gredo skozi različne stopnje v času učenja in privajanja na projektu [23]:

- kot **novinec** (ang. newbie); oseba, ki sistem šele odkriva, spoznava,
- kot **vmesni uporabnik** (ang. intermediate); uporabnik z zadostnim znanjem (ang. know-how) za uporabo sistema in nadaljno učenje,
- kot **napreden uporabnik**; oseba, sposobna reševati probleme drugih in vključena v promocijo sistema,
- kot **ekspert**; oseba, ki ima zadnjo besedo pri zadevah na projektu in globoko tehnično znanje o funkcionalnostih sistema.

Ljudje v kontekstu odprtokodnega projekta medsebojno vpivajo in vzajemno delujejo na različne načine:

- s pisanjem, spreminjanjem in razhroščevanjem dejanske izvorne kode,
- s poročanjem in komentiranjem hroščev,
- s pisanjem in branjem dokumentov kot so uporabniški priročniki, sistemske dokumentacije, tehnični priročniki itd.,
- z javnimi diskusijami preko poštnih seznamov, novičarskih skupin (ang. newsgroups), IRC sej itd.,
- z obiskovanjem uradne spletne strani projekta in povezanih spletnih strani,
- z udeleževanjem na sestankih in konferencah kot so uporabniške skupine (ang. user groups), delovne skupine (ang. working groups), sestanki skupnosti (ang. community meetings) itd.

Uporabniki lahko razpravljajo, kako najbolje opraviti svoje delo. Pogovor je lahko osredotočen na standarde za uporabljene protokole, API vmesnike[43], ki jih program uporablja, funkcije in procedure programa ali katero drugo področje projekta. Vsaka posamezna razprava vključuje različne skupine ljudi. Te skupine lahko obsegajo le nekaj ljudi, ali pa so to kar skupnosti s sto ali več tisoč sodelujočimi posamezniki. Te mnogovrstne skupnosti prikazuje slika 2.6.




**Slika 2.6 Skupnosti na podlagi skupnih interesov**

[43] API – Application Program Interface: vmesnik, ki združuje množico procedur, protokolov in orodij za izgradnjo aplikacij

Nekateri posamezniki se bodo udeleževali različnih razprav, s čimer bodo postali člani več skupnosti. Tako prihaja do prekrivanja med njimi (skupnostmi), kar omogoča ustanovitev večje skupnosti, povezane z odprtokodnim projektom kot celoto. Vsaka se bo okrepila ali pa oslabela glede na to, kako dobro bodo njeni interesi podprti z njenimi viri. Na primer skupnost novincev, ki povprašuje o tem, kako uporabiti določen del programa bo uspešna le, če bodo bolj izkušeni uporabniki, ki lahko odgovorijo na njihova vprašanja, prav tako člani te skupnosti [26].
Glede na stopnjo vpletenosti oziroma sodelovanja pri projektu, ločimo naslednje kategorije uporabnikov:

- prežalec (ang. lurker) (najmanjša stopnja vpletenosti),
- darovalec (ang. contributor),
- razvijalec (ang. core developer) (največja stopnja vpletenosti).

Prežalci »prisluškujejo« na poštnih ali diskusijskih seznamih (ang. discussion list), kjer ne puščajo svojih sledi. Ne objavljajo ničesar in ne sodelujejo v razpravah, temveč so prisotni samo kot bralci. Tako so za vse ostale člane projekta na nek način nevidni, vendar imajo svojo vlogo pri projektu. Pomagajo promovirati standarde z uporabo programa, ki se razvija, pomagajo širiti sloves projekta in predstavljajo potencialne nove osrednje razvijalce, saj so tudi ti začeli z branjem poštnih seznamov in preizkušanjem programa.
Drugo kategorijo predstavljajo (običajni) darovalci, ki so največja vidna skupina članov projekta. Darovalci sodelujejo v diskusijah, objavljajo sporočila v poštnih seznamih in prispevajo programsko kodo preko »čuvaja vrat« (razvijalca), ki ima pisalni (ang. write) dostop do CVS sistema in ovrednoti ter preuči ustreznost prispevka darovalca.

Stopnja vpletenosti je praviloma povezana s tehničnim znanjem, zaslugami in občutkom odgovornosti. Razvijalci prispevajo večino programske kode, planirajo objavo verzij in odločajo o vključevanju značilnosti programa (ang. features) ali raznih odprtih zadev, ki vplivajo na splošno usmerjenost razvoja. Gledano s tehničnega vidika, so osrednji razvijalci vsi tisti, ki imajo CVS pisalni dostop. CVS hrani najbolj ažurne verzije programa in dovoljuje več uporabnikom hkrati spreminjati kodo. Tak dostop je dovoljen samo relativno majhnemu krogu kvalificiranih, izkušenih programerjev, ki so pridobili zaupanje drugih razvijalcev s svojimi prispevki in trudom [6].

Če se bolj podrobno osredotočimo na vloge posameznikov v odprtokodnem projektu, opazimo naslednje vloge:

**Razvijalec (ang. Developer)**

Razvijalec je zadolžen za implementiranje ciljev projekta. Udeležuje se predvsem produkcijskih in dokumentacijskih aktivnosti.

**Upravljalec (ang. Manager)**

Upravljalev upravlja, vodi projekt. Sprejemanje odločitev in koordinacija sta njegovi glavni aktivnosti.

**Vzdrževalec (ang. Maintainer)**

Vzdrževalec beleži probleme izdanih komponent in je zadolžen za njihovo reševanje. Njegovi glavni aktivnosti sta koordinacija in komunikacija.

**Administrator**

Administrator je odgovoren za pravilno delovanje projekta z vzdrževanjem projektnih virov, kot so centralizirani strežniki, komunikacijske zveze, sistemi za upravljanje konfiguracij.

**Poročevalec (ang. Commenter)**

Vsakdo, ki podaja povratne informacije je poročevalec.

### 2.5.3 Življenski cikel projekta

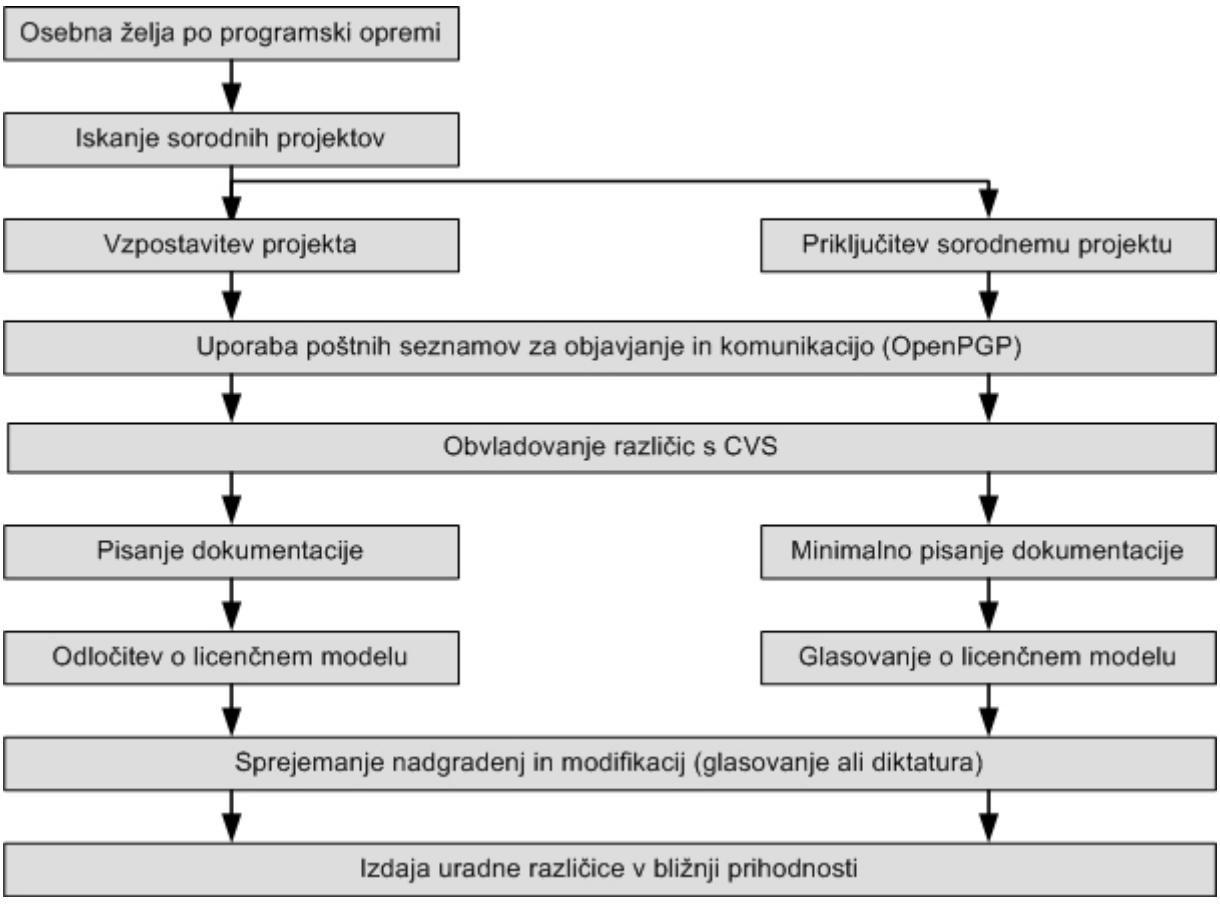
Odprtokodni projekti so organski projekti. Ne sledijo strogim načelom, ki bi napovedovala izdajo verzij in oscilirajo med posameznimi fazami življenskega cikla. Tipičen življenski cikel izgleda takole:

1. Nekdo ima nek nerešen problem, potrebo po programu in skuša najti rešitev.[44]
2. Oseba povpraša prijatelje in kolege, če kaj vedo o njegovem problemu. Nekateri od njih utegnejo imeti podobne probleme in so verjetno prav tako še brez rešitve.
3. Vse zainteresirane osebe začnejo izmenjevati znanje o tej temi in si s tem začrtajo bledo, megleno sliko o osrednjem problemu skupine.
4. Zainteresirani ljudje, ki so pripravljeni potrošiti nekaj svojih virov (čas, znanje) za iskanje rešitve problema, vzpostavijo neformalen projekt.
5. Člani projekta se ukvarjajo s problemom dokler ne dosežejo določenih zadovoljivih rezultatov.

[44] Eric Raymond je to poimenoval s stavkom: »*Every good work of software starts by scratching a developer's personal itch.*«

6. Svoje dosedanje delo javno objavijo na nekem mestu, kjer je omogočen dostop veliki množici ljudi. Svoje projekte lahko oznanijo na mestih, kot so poštni seznami, novičarske skupine itd.
7. Druge osebe spoznajo nekatere svoje skrbi in interese v projektu in so prav tako zainteresirane za primerno rešitev. Zato preučijo trenutne rezultate projekta (npr. z njegovo uporabo). Ker gledajo na problem z drugačne perspektive, predlagajo morebitne izboljšave in se utegnejo celo pridružiti projektu.
8. Projekt se širi in veliko povratnih informacij pripomore k boljšemu razumevanju problema ter pripelje do možnih strategij za izboljšavo rešitve problema.
9. Nove informacije in viri se integrirajo v raziskovalni proces. Rešitev se širi in v vedno večji meri pokriva prvoten problem.
10. Razvojni cikel se zaključi in se vrne v fazo 5.
11. Projektna skupnost je osnovana in bo reagirala na bodoče spremembe na enak način, kot je v začetku.

Ta proces v bolj poenostavljeni obliki prikazuje slika 2.7.




**Slika 2.7 Življenski cikel odprtokodnega projekta (Vir: [27])**

Splošna klasifikacija različnih stopenj razvoja odprtokodnih projektov je Načrtovanje, Pred-Alfa, Alfa, Beta, Stabilna, Zrela [9].

**Načrtovanje (ang. Planning)**

Programska koda še ni bila pisana, področje projekta je še nedefinirano. Projekt je zaenkrat samo ideja. Takoj, ko se pojavijo otipljivi rezultati v obliki izvorne kode, projekt preide v naslednjo fazo.

**Pred-Alfa (ang. Pre-Alpha)**

Začetna različica izvorne kode je bila izdana. Ne pričakuje se, da bi se koda prevedla (ang. compile) ali izvršila. Zunanji opazovalci utegnejo imeti težave z razumevanjem kode. Šele, ko je viden nek skladen načrt v kodi, ki nakazuje dokončno smer, projekt preide v naslednjo fazo.

**Alfa (ang. Alpha)**

Izdana koda vsaj začasno deluje in začenja dobivati ustrezno obliko. Pojavljajo se začetne razvojne beležke in zapiski. Aktivno delo na širjenju funkcionalnosti aplikacije se nadaljuje. Ko se količina novih funkcij aplikacije umiri, projekt preide v naslednjo fazo.

**Beta**

Programska koda obsega vse željene funkcionalnosti, vendar še vsebuje napake. Le-te so postopoma odpravljene, kar vodi do izdelka, ki je vedno bolj zanesljiv. Če je število napak zmanjšano na ustrezno število, projekt izda stabilno verzijo in preide v naslednjo fazo.

**Stabilna (ang. Stable)**

Program je uporaben in dovolj zanesljiv za dnevno uporabo. Spremembe so vključene zelo pazljivo in njihov namen je povečati stabilnost, ne dodajanje funkcionalnosti. Če se dalj časa ne pojavi potreba po večjih spremembah in ostajajo nerešene le manjše zadeve, projekt preide v naslednjo fazo.

**Zrela (ang. Mature)**

Razvoja je zelo malo, ali pa ga sploh ni, saj program izpolnjuje svoj namen zelo zanesljivo. Spremembe se izvajajo z izjemno pazljivostjo ali pa sploh ne. Projekt ostane v tej fazi več let,

dokler ne zastara. Izvorna koda ostaja dostopna ves čas in lahko služi za izobraževalne namene.

## *2.6 Prednosti in slabosti prostega programja*

### 2.6.1 Prednosti[45]

Zagovorniki prostega programja pogosto trdijo, da prosto programje ponuja nekaj bistvenih prednosti v primerjavi s tipičnimi komercialnimi produkti. Komercialni produkti večinoma dajejo prednost izgledu, torej vizualnim podrobnostim, pred težje merljivimi kvalitetami kot so stabilnost, varnost in drugim manj »glamuroznim« atributom.

Za razvijalce prostega programja naklonjenost posebnostim (ang. features) pred kvaliteto prav gotovo ni značilna. Za njih je značilna predvsem želja po izpopolnjevanju znanja ter iskanju samopotrditve in ugleda med kolegi v odprtokodnih skupnostih. Zato je bolj verjetno, da bodo pri razvoju programja dali poudarek stvarem, ki bodo s strani somišljenikov cenjene. To so predvsem jasen načrt (ang. design), zanesljivost, enostavno vzdrževanje s poudarkom na splošnih, odprtih standardih in vrednotah skupnosti.

**Zanesljivost**

Zanesljivost v grobem pomeni odsotnost napak v programu, ki bi povzročile nepravilne operacije, izgubo podatkov ali nenadne izpade programa.

Osnovno načelo odprtokodnih projektov je »*izdajaj zgodaj, izdajaj pogosto*« (ang. release early, release often). Pogoste izdaje verzij programa omogočajo pridobivanje kvalitetnih povratnih informacij zelo različnih uporabnikov, prav tako se hitreje odkrijejo napake. Te so običajno popravljene v nekaj urah, kar je predvsem posledica dostopnosti izvorne kode programa. Sposobni razvijalci, ki napako odkrijejo, jo običajno kar sami tudi popravijo in nato obvestijo vzdrževalca. Po možnosti izdajo svojo popravljeno verzijo programa. Uporabniki se tako lahko odločijo, ali bodo uporabili neuraden popravek, ali bodo počakali na uradno verzijo.

**Stabilnost**

V poslovnem okolju je programska oprema nujno zlo, orodje za delo. Če v procesu dela ni večjih sprememb tudi ni potrebe po spreminjanju, nadgrajevanju programske opreme. Lahko celo rečemo, da se uporabniki zaradi stroškov spremembam izogibajo, kar pa je v nasprotju z motivi proizvajalcev programske opreme, ki si želijo stalnega pritoka dohodkov. V ta namen

[45] povzeto po [28] in [29]

uporabljajo različne taktike, s katerimi želijo svoje uporabnike prepričati ali prisiliti v spremembe ali nadgradnje (npr. podpora novim formatom, umik tehnične podpore in popravila hroščev za starejše verzije ali starejše platforme). Uporabniki so tako na nek način prepuščeni na (ne)milost proizvajacev.

Prosto programje je prilagojeno odprtim standardom, ti pa se redko spreminjajo. Zato spremembe v programski opremi niso pogosto potrebne in se uporabnike ne sili v njih. Nekompatibilnost zato ni problem. Uporabnikom nudi možnost, da se odločijo, ali bodo, če za nadgradnjo ni potrebe, ostali na starejši verziji ali bodo prešli na novo verzijo programa.

**Preglednost (ang. Auditability)**

Redko razumljena prednost odprtokodnega programja je njegova preglednost. Pri lastniškem programju so uporabniki prisiljeni v zaupanje trditvam razvijalcev o kvalitetah kot so varnost, neobstoj stranskih vrat (ang. backdoor), prilagojenost standardom in fleksibilnost za bodoče spremembe. Če izvorna koda ni dostopna, so to le obljube.

Z objavo izvorne kode avtorji vlijejo zaupanje uporabnikom, da so takšne in podobne trditve na trdnih temeljih, saj se o tem lahko sami prepričajo. Če izvorna koda ni objavljena, nadzor s strani tretje osebe (specialne institucije) ni možen. Trenutno industrija sicer tega ne zahteva, vendar se s povečevanjem razširjenosti odprtokodnega modela v prihodnje to lahko uveljavi.

**Stroški**

S poslovnega vidika so najpomembnejši pokazatelji stroškov produkta tako imenovani skupni stroški lastništva - TCO (Total Cost of Ownership). Pri izbiri med več možnimi rešitvami, pod predpostavko, da so vsi ostali kriteriji izenačeni, je rešitev z najmanjšim TCO najbolj zaželjena. Argumenti, ki govorijo v prid nizkemu TCO za prosto programje so[46]:

- Po vsej verjetnosti nični stroški nakupa oziroma pridobitve programa in posledično ni stroškov obnavljanja licenc uporabe programa.
- Potencialno ni dodatnih stroškov za uporabo kopij programa, programe lahko brez dodatnih stroškov prenesemo na več računalnikov.
- Manjša potreba po nadgradnjah (manjši stroški nadgradenj in upravljanja), hkrati so nadgradnje programa večinoma brezplačne.

[46] avstralsko podjetje Cybersource je sredi decembra 2004 objavilo študijo o skupnih stroških lastništva programske opreme. Študija je pokazala, da podjetje z 250 zaposlenimi lahko prihrani do 36 odstotkov s prehodom na odprtokodni operacijski sistem in programsko opremo, ki teče na njem. Povezava do študije: http://www.cybersource.com.au/about/linux_vs_windows_tco_comparison.pdf

- Daljše delovanje brez izpadov in posledično nižji stroški za drage sistemske administratorje.
- Nižja ranljivost na viruse, kar izloči potrebo po pregledovanju za virusi, zmanjšuje možnosti izgube podatkov.
- Nižja ranljivost do varnostnih lukenj in hekerskih napadov, kar znižuje breme sistemske administracije.
- Možnost uporabe starejše oziroma manj zmogljive strojne opreme ob enakih učinkovitostih, kar znižuje stroške strojne opreme.

Zgornje trditve veljajo predvsem za večje odprtokodne projekte. Pri marsikaterem manjšem odprtokodnem projektu bi nekatere od njih naletele na krhke temelje.

**Fleksibilnost in svobodnost**

V poslovnem kontekstu fleksibilnost programske opreme pomeni možnost izbire rešitve, ki bo zadovoljila vse potrebe uporabnikov. V primeru sprememb v poslovnem okolju, le-te niso omejene s programom, ki podpira poslovanje. Predvsem je to pomembno na področju infrastrukturnih komponent – arhitekture IT rešitev. Izkušnje kažejo, da je za zagotovitev fleksibilnosti na arhitekturnem nivoju najbolje izbrati preizkušene standarde za vzajemno delovanje (ang. interworking)[47]. Pod pogojem, da lahko posamezne rešitve (sistemi, programi) ustrezno vzajemno delujejo, se je možno izogniti preveliki odvisnosti od določenega proizvajalca programske opreme.

Odprtokodni projekti se v ta namen poslužujejo predvsem de iure[48] in de facto[49] standardov. Zato pri prostem programju ni zaslediti odvisnosti od sorodnih produktov[50]. Prosto programje

[47] interworking pomeni, da sistemi ali komponente različnih proizvajalcev lahko vzajemno delujejo pri opravljanju določene naloge. Standardi definiranje vmesnikov med komponentami so odločilni za uspešno vzajemno delovanje. Izraz upošteva, da obstajajo določene razlike med komponentami, ki bi bile ob odsotnosti splošnih standardov nezmožne skupaj delovati.

[48] de iure standardi nastajajo po zakonu pod pokroviteljstvom avtoriziranih agencij za standardizacijo. Nastajanje takšnega standarda je zaprto v toge strokovno administrativne postopke, ki imajo za posledico veliko počasnost pri sprejemanju standardov.

[49] de facto standardi nastajajo neodvisno od mednarodno priznanih organizacij za standardizacijo: VMS, DOS, UNIX, PC. Rojevajo se veliko bolj praktično in se potrjujejo s svojo komercialno in uporabniško kvaliteto.

[50] dejstvo, da se pri razvoju prostega programja uporabljajo predvsem odprti standardi, odpravlja bojazen, da rezultatov programja v obliki različnih formatov datotek ne bi bilo možno uporabiti v drugih programih, ki prav tako temeljijo na odprtih standardih. Odprti standardi so javno dostopne specifikacije za izvršitev določene naloge. Nasprotno se proizvajalci lastniškega programja običajno poslužujejo zaprtih standardov (specifikacije niso javno dostopne), s čimer drugim onemogočajo razviti programje, ki bi lahko uspešno uporabljalo ali obdelovalo rezultate lastniškega programja. Uporabniki so tako za nadaljno uporabo ali obdelavo rezultatov določenega programja primorani uporabiti rešitev istega proizvajalca.

ponuja svojim uporabnikom večjo svobodo pri nakupu oziroma uporabi drugih produktov, izogibajoč se odvisnosti od določenega proizvajalca.

Proizvajalci programske opreme lahko prenehajo poslovati, ali se samovoljno odločijo, da bodo prenehali z razvojem določenega produkta. Kaj lahko neko podjetje, ki je odvisno od nadgradenj in nadaljnega razoja programske opreme določenega proizvajalca, v takem primeru stori? Glede na to, da je lastniško programje zaščiteno z avtorskimi pravicami, ga nobeno drugo podjetje ne more prevzeti in nadaljevati z razvojem. Prosto programje učinkovito ščiti podjetja pred takšnimi in podobnimi scenariji. Pogosto je možno najeti neko drugo skupino programerjev, ki bo nadaljevala z vzdrževanjem in nadaljnim izboljševanjem programske opreme, brez zakonskih omejitev.

Odprtokodni projekti pogosto omogočajo bolj fleksibilno kombinacijo posebnosti (ang. features) programja. Običajno dela veliko ljudi na svojih verzijah določenega programa, tako da je popolnoma možno najti verzijo, ki zadovolji vse potrebe določenega uporabnika. Če ne obstaja verzija programa, ki bi dovolj dobro ustrezala, jo je glede na odprtost programske kode možno samostojno izdelati, ali pa najeti nekoga drugega za to.

**Podpora**

Razvoj z odprto kodo lahko nudi zelo hitro in kvalitetno tehnično podporo. Glede na to, da uporabniki in razvijalci sodelujejo na osnovi svojih skupnih interesov in potreb po določeni aplikaciji, so razvijalci praviloma pripravljeni vložiti nekaj dodatnega napora za pomoč ostalim. Odgovore na vprašanja je možno dobiti na različnih mestih, kot so novičarske skupine, poštni seznami, klepetalnice itd.

### 2.6.2 Slabosti (problemi) odprte kode

**Kompatibilnost**

Prosto programje ne deluje vedno najbolje z drugimi aplikacijami, tipično z Microsoft aplikacijami kot je Windows operacijski sistem (ne uporabljajo dovolj dobro značilnosti Windows okolja in ga tako ne uspejo maksimalno izrabiti).

**Uporabniški vmesniki**

Prosto programje ima običajno manj prijazne uporabniške vmesnike. To je predvsem problem pri uporabnikih, ki niso vešči dela v računalniškem okolju in jim neprijazni uporabniški vmesniki še bolj otežijo delo. Pogosto vmesniki niso dovolj prilagojeni nezahtevnim

uporabnikom. Glavni vzrok za to je, da je prosto programje večinoma grajeno s strani inženirjev za inženirje, saj pravih povratnih informacij nezahtevnih končnih uporabnikov razvijalci ne dobijo. Nezahtevni končni uporabniki namreč ne sodelujejo v procesu razvoja odprtokodne aplikacije, saj nimajo potrebnega znanja za testiranje aplikacije in učinkovito poročanje o hroščih. Posledica tega so bolj programersko usmerjeni vmesniki kot pa uporabniški vmesniki[51].

**Podpora**

Dobra in raznovrstna podpora je bila omenjena že kot prednost, vendar je potrebno dodati, da je podpora včasih težko razumljiva, saj je pogosto bolj namenjena razvijalcem in ne končnim uporabnikom. Običajno gre za zelo tehnično podrobno dokumentacijo, ki zahteva veliko tehničnega predznanja. Poleg tega je večinoma tudi zelo pomanjkljiva. Njen največji del predstavlja kar izvorna koda, na podlagi katere je avtomatsko generirana programska dokumentacija. Preostala dokumentacija je pogosto neažurna, zaradi pogostih izdaj programja.

**Namestitev aplikacije**

Prosto programje pogosto ni tipa »vstavi in poženi« (ang. plug and play). Čeprav se odprtokodni projekti vedno bolj ukvarjajo z vprašanjem enostavnosti uporabe, ostaja namestitev aplikacije še vedno velika ovira za veliko uporabnikov. Niso podprti vsi gonilniki naprav in pogosto sam proces nameščanja programja vodi uporabnika skozi zaporedje vprašanj, o katerih uporabnik ne želi razmišljati, ali pa o njih sploh nič ne ve.

**Intelektualna lastnina in pomanjkanje lastništva (odgovornosti)**

Komercialna podjetja programske opreme si zelo prizadevajo zagotoviti, da programska oprema, ki jo razvijajo, ne vsebuje kršitev intelektualne lastnine. Od zaposlenih se zahteva, da v svoje komercialne programe ne prepisujejo posameznih delov programske kode, ki so jih napisali posamezniki izven podjetja.

Nasprotno je zaradi narave odprtokodnih projektov, kjer sodeluje veliko samostojnih razvijalcev, večja verjetnost, da obstajajo deli programja, kjer se krši pravica intelektualne lastnine. Samostojen razvijalec morda vključi v prosto programje določen del programske

[51] primeri bolj znanih produktov prostega programja s »programerskimi« uporabniškimi vmesniki: Apache, Perl, PHP.

kode lastniškega programa, ki ga je v nekem komercialnem podjetju razvijal. Posledično je uporaba takšnih produktov za uporabnike bolj tvegana.

Uporabniki želijo, da nekdo prevzame odgovornost. Ta pa je pri podjetju lastniškega programja jasneje določena kot pri prostem programju. Lastniška podjetja lahko pred potencialnimi in trenutnimi uporabniki nastopajo kot zaupanja vredni skrbniki. Microsoft se v ta namen poslužuje taktike Strah (ang. Fear), Negotovost (ang. Uncertainty), Dvom (ang. Doubt). FUD je tržna taktika, ki jo uporabljajo predvsem podjetja z velikim tržnim deležom, ker niso sposobna s trdnimi dejstvi odgovoriti konkurenčnim produktom, ki so boljši in cenejši. Taktika ustvarja govorice in dvom pri potencialnih uporabnikih, češ »Tvegano je iti po tisti poti, raje ostanite z nami in boste z množico. Naša naslednja verzija bo tako ali tako boljša.«

V nasprotju s prostim programjem lahko lastniška podjetja zajamčijo kompatibilnost s starejšimi verzijami ter s tem predstavljajo entiteto, ki jo je možno tožiti v primeru, da njihove trditve ne držijo.

**Težak začetek in težko zaključevanje**

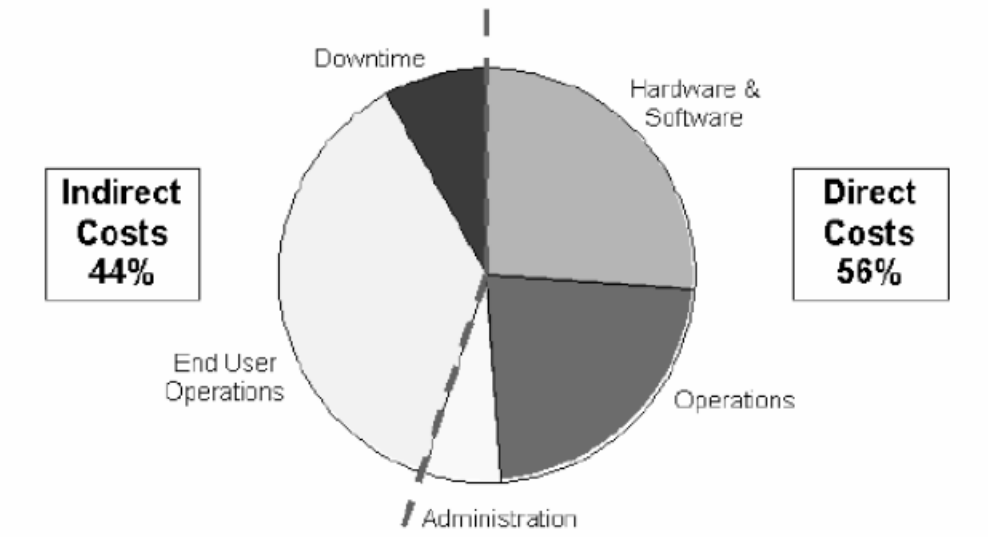
Pogoj za to, da odprtokodni projekt zaživi je, da mora privabiti dovolj veliko skupnost zelo izurjenih in zainteresiranih razvijalcev, usmerjenih v rešitev problema projekta. Večji kot je projekt, več je razvoja in razhroščevanja kode. Razvijalci morajo imeti skupen cilj, ki je jasno in dobro definiran ter analogen poslanstvu skupnosti. Veliko odprtokodnih projektov težko pridobi veliko skupnost razvijalcev in drugih uporabnikov. Pri takih projektih je razvoj bolj podoben »katedralnemu« kot pa »bazarnemu«. Veliko odprtokodnih projektov ima težave pri prehodu v stabilno oziroma zrelo stanje, saj se zanimanje za razvoj projekta pogosto prej konča. Tako veliko odprtokodnih projektov nikoli ne zapusti beta faze razvoja.

## *2.7 Ekonomski vidik*

Odprtokodni projekti močno vplivajo na ekonomski vidik programske opreme. Zavedati se je potrebno, da stroški nakupa ne predstavljajo samo stroškov, ki so povezani s programsko opremo, ampak jih tvorijo tudi naslednji dejavniki [9]:

- **priprava sistema**; za zagon novega programa je običajno potrebno konfigurirati številne nastavitve in komponente (strojna oprema, infrastruktura).

- **učinkovitost izvajanja**; vsak program zahteva v času svoje življenjske dobe določena dodatna sredstva, kot so človeški viri in čas, potreben za interakcijo s programom. Učinkovitost sredstev pri izvajanju operacij močno vpliva na stroške.
- **odpovedi**; odpoved programa je lahko povezana z visokimi stroški.
- **izobraževanje**; večina programov je tako kompleksnih, da zahtevajo dodatno usposabljanje, bodisi s pomočjo dokumentacije, tečajev ali neposrednega učenja pri delu s programom.
- **storitve**; vključujejo stroške za podporo, »helpdesk« in podobne procese.
- **nadgradnje**; nadgradnje programov odpravljajo napake in dodajajo nove funkcionalnosti, kar pa pogosto ni brezplačno.
- **nakup**; predstavlja dovoljenje za uporabo kopije programske opreme, ki običajno ne zagotovlja tudi lastništva.




**Slika 2.8 Delež stroškov povezanih z nakupom programske opreme**

### 2.7.1 Oddajanje programske opreme brezplačno

Eno izmed ekonomskih gonil, zakaj se odreči zaslužku pri oddajanju (odprtokodne) programske opreme je, da se z njenim plasiranjem na trg okrepi in vzdržuje tržni delež za lastniško programsko opremo, ki jo organizacija trži. Na primer, odprtokodna odjemalska aplikacija, ki se ponuja brezplačno, pospešuje prodajo ustrezne strežniške aplikacije. Na tem

konceptu temelji ekonomski model »Loss-Leader«[52], ki so ga uspešno uporabili pri podjetju Netscape Communications, Inc. S tem, ko so izdali odprto kodo svojega Netscape brskalnika, so preprečili Microsoftu, da bi prišel do monopola na tem področju. Prav tako se je delež uporabnikov Netscape-a drastično povečal.

### 2.7.2 Zaslužek z odprto kodo

Uspešni ekonomski modeli za trženje odprte kode temeljijo na naslednjih konceptih:

- **Distribucija programske opreme**; Distributorji preprosto tržijo kopije odprtokodne programske opreme. To poslovanje temelji na ideji, da so običajni uporabniki pripravljeni plačati nek minimalen znesek za ustrezen dostop in uporabo programske opreme. Pri takšnem poslovanju se trži celoten paket, torej kopija programske opreme z vso pripadajočo literaturo in dokumentacijo ter še kakšnim dodatkom. Paketi so zaščiteni z blagovno znamko in avtorskimi pravicami.
- **Storitve**; kot so odpravljanje napak, izobraževanje ali podpora, se lahko tržijo. Na tem konceptu temelji ekonomski model »Support Sellers«[53], kjer se postavi odprtokodno programsko opremo na trg z namenom okrepiti tržni delež njenih storitev. Ta model uporabljajo pri Red Hat[54] in nekaterih drugih distributerjih Linux operacijskega sistema.
- **Prodaja strojne opreme**; Glede na to, da strojna oprema ni uporabna brez ustrezne programske opreme, so proizvajalci na nek način prisiljeni razvijati in vzdrževati programsko opremo (gonilnike in različne programe, ki izrabljajo funkcionalnosti strojne opreme). Običajna praksa je, da se ta programska oprema ponuja brezplačno, vendar brez izvorne kode. Tak razvoj in vzdrževanje programske opreme ne prinaša prihodkov, temveč samo ogromne stroške. Tako je vse več in več podjetij začelo sodelovati pri odprtokodnih projektih, da bi zagotovili kompatibilnost in podporo svojih produktov. S prehodom na odprto kodo takšna podjetja pridobijo veliko zaledje razvijalcev, hitrejše in fleksibilnejše odzivanje na potrebe uporabnikov ter večjo zanesljivost. Na tem konceptu temelji ekonomski model »Widget Frosting« in je

[52] izraz loss-leader se uporablja za izdelek, ki se prodaja po ceni, ki je nižja od stroškov njegove izdelave z namenom privabiti kupce k svojim drugim izdelkom. Pričakuje se, da bo tipičen uporabnik poleg »loss-leader« izdelka hkrati kupil tudi kakšne druge izdelke in, da bodo prihodki s prodajo drugih izdelkov tolikšni, da bodo pokrili izgubo, ki jo prinašajo »loss-leader« izdelki oziroma bo v celoti gledano pridelan dobiček.
[53] Ta ekonomski model je znan tudi pod imenom »Izdaj recept, odpri restavracijo« (ang. Give away the recipe, open a restaurant)
[54] http://www.redhat.com

predvsem primeren za proizvajalce strojne opreme. Najbolj znana primera podjetij, ki sta ubrala to pot sta Nvidia inc.[55] in Adaptec inc.[56]

- **Dodatki**; tržijo se lahko dodatki odprtokodne programske opreme. Trži se vse, od majic in drugih artiklov s simboli odprtokodne programske opreme (npr. simbol pingvina za Linux), do knjig, revij ali celotnih sistemov z nameščenim prostim programjem.

## *2.8 Uporaba odprte kode*

Uporabniki (poslovni in domači) programske opreme imajo danes širok spekter možnosti za zadovoljitev svojih računalniških in omrežnih potreb. Ena od teh možnosti je tudi prosto programje, ki je med uporabniki vse bolj pogosto izbrano, saj običajno ponuja več za svojo ceno. Tipični primeri uporabe prostega programja pri domačih uporabnikih in v podjetjih so:

- **uporaba prostega programja kot zamenjava ali nadomestilo za komercialno (lastniško) programsko opremo;** Obstaja veliko prostega programja, ki opravlja enake funkcije kot njihovi »lastniški bratje«. Najbolj znani primeri so uporaba operacijskega sistema Linux namesto MS Windows, uporaba OpenOffice namesto MS Office, uporaba spletnega brskalnika Mozilla namesto brskalnika MS Internet Esplorer itd.
- **odprtje obstoječega (lastniškega) programja**; Pri podjetjih, ki se odločijo za odprtje lastnega produkta, to je za objavo izvorne kode produkta, gre v ozadju običajno za konkurenčno grožnjo. Njihov namen je prevetriti trg, zagreniti življenje večjim konkurentom. Taka poteza podjetjem omogoča zmanjšati breme vzdrževanja produkta, s tem ko privabijo ljudi k sodelovanju. Včasih se odločijo za objavo izvorne kode svojega produkta ali določene tehnologije z namenom, da bi jo lažje tržili. Tako je IBM v operacijski sistem Linux vključil tehnologiji XFS[57] in JFS[58]. Znani so tudi primeri objave izvorne kode produkta zaradi njegove zastarelosti ali prenehanja donosnosti[59].

[55] http://www.nvidia.com
[56] http://www.adaptec.com
[57] visoko-performančni datotečni sistem za dnevnike (ang. journaling file system), ki ga je razvil SGI (Silicon Graphics Inc.)
[58] visoko-performančni datotečni sistem za dnevnike (ang. journaling file system), ki ga je razvil IBM
[59] v svetu iger najbolj znana primera odprtje pogonov (ang. engine) iger Quake 1 in Doom, kar je omogočilo razvoj raznih nadgrajenih oziroma modificiranih verzij (ang. mod) iger, ki so ponujale določene spremembe in dodatke. S tem se je zanimanje za te igre zopet povečalo, posledično so bili večji tudi dohodki.

- **Grajenje okoli obstoječega prostega programja;** Gre za uporabo obstoječega prostega programja, ki s svojimi funkcijami dokaj dobro zadovoljuje potrebe uporabnika. Z nadaljnim razvojem takega programja, torej z dodajanjem in spreminjanjem obstoječe izvorne kode, z njegovo profesionalizacijo, ga je možno popolnoma prilagoditi potrebam uporabnika.
- **uporaba odprtokodne programske opreme kot izhodišče za razvoj novega programa;** Del kode iz obstoječega odprtokodnega projekta se lahko uporabi za začetek procesa razmišljanja o razvoju (po možnosti) povsem drugačnega projekta. V tem primeru je možnost vpogleda v izvorno kodo katalizator novih idej in pristopov, ki bi jih bilo moč uporabiti pri drugem projektu. V nekaterih primerih se manjša količina izvorne kode izposodi za začetek dela na novem projektu in se nato v procesu razvoja tega projekta spremeni do te mere, da ni več nobene podobnosti z njenim prvotnim stanjem. S tem se neposredno izogne morebitnim kršitvam pravil prava intelektualne lastnine.
- **razvoj novega prostega programja;** Za podjetja, ki se odločijo za razvoj nove odprtokodne rešitve, pomeni to vstop na nov trg, kjer je možno hitrejše pridobivanje potrebnega znanja (ang. know-how) za razvoj željenega produkta in večjo promocijo produkta ter podjetja. Tak razvoj programja je cenejši, saj je mogoče izkoriščati delo prostovoljcev, ki med drugim posredujejo popravke in izboljšave ter opravljajo testiranje produkta. Hkrati je možno tudi sodelovanje s podjetji s podobnimi zahtevami, ki niso neposredni konkurenti (npr. zaradi delovanja na dugem kontinentu). Proces razvoja mora biti organiziran tako, da je vstopanje vanj enostavno (uporaba dopisnih seznamov, ažurirana spletna stran, upravljanje z verzijami itd.).
- **dodajanje zaprtokodnih manjkajočih členov;** Tudi v tem primeru je izhodišče obstoječe (običajno široko uporabljeno) prosto programje. Podjetje se lahko odloči za zaprto izboljšavo ali razširitev takšnega produkta. Takšna odločitev prinaša nižje stroške razvoja, saj osnovo končnega produkta razvijejo drugi. Začetni trg končnega produkta je velik, saj so njegovi potencialni uporabniki vsi obstoječi uporabniki prostega programja, ki se razširja. Predpogoj za vse to je, da licenca prostega programja dovoljuje, da spremembe ostanejo zaprte in da se ne širijo naprej.

V nadaljevanju bom predstavil odprtokodno aplikacijo Bugzilla in način njene uporabe v podjetju Comland d.o.o.

# 3 COMLAND D.O.O. IN BUGZILLA

Comland d.o.o. je podjetje, specializirano za razvoj informacijskih rešitev. Pri podjetju se gradi na prepričanju, da je za uspeh projekta potrebno dobro sodelovanje med izvajalcem in naročnikom. Stranka investira z željo izboljšati svoje poslovanje; njen cilj je zaslužiti na dolgi rok. Da bi zaščitili njene interese, je potreben predvidljiv delovni proces. Proces izvedbe projekta mora biti v vsakem trenutku transparenten. Projekt mora biti vseskozi pod nadzorom; kadarkoli mora biti možno pregledati dogajanje na njem.

Da bi sodelovanje z naročniki še izboljšali, transparentnost in organiziranost dela pa še povečali, smo se odločili za razvoj helpdesk[60] aplikacije, ki bi naše delo in storitve še bolj približala naročnikom. Helpdesk aplikacija naj bi bila namenjena uporabnikom naših programov (programov, ki jih vzdržujemo), kjer bi lahko podajali svoje težave pri delu s programi ali morebitne napake, na katere so naleteli pri svojem delu s programom.

## *3.1 Specifikacija helpdesk aplikacije*

V začetni fazi izdelave helpdesk aplikacije je bilo potrebno opredeliti naše zahteve in predvsem odgovoriti na vprašanja tipa: kakšen je namen aplikacije, kaj od nje pričakujemo, katere funkcije naj podpira, ipd. Projekt smo definirali s poslovnega stališča in določili njegov obseg.

### 3.1.1 Zahteve

Naše zahteve glede helpdesk aplikacije so bile:

- da je možno prijavo napake ali predloga podati kadarkoli
- da je možno vedno pogledati, v kateri fazi je reševanje prijave
- da je za vsako napako možno preveriti, kaj se je že zgodilo v teku njenega reševanja (sledljivost)
- da se dokumentacija prijave in obvestila čim hitreje in po možnosti samodejno predajajo naslednjim nivojem
- da se nobena prijava ne »izgubi«, torej da se vse rešijo, četudi kakšna neuspešno

[60] informacijska podpora uporabnikom, kjer lahko najdejo rešitve svojih problemov s programsko ali strojno opremo.

- da je mogočih več vlog v aplikaciji, od katerih ima vsaka svoje pravice, ter da se lahko osebe, ki pripadajo določeni vlogi, enostavno nadomesti z drugimi in se pri tem ne izgubi sledljivost

### 3.1.2 Funkcionalnost

Rešitev mora vsebovati naslednje funkcionalnosti:

- prijava napak in predlogov
- pregled prijav
- posredovanje prijav v reševanje
- reševanje prijav
- analiza podatkov o prijavah
- zbirka znanja[61]

**Vloge v aplikaciji**

Vloge v aplikaciji bodo nastavljive, kar pomeni, da jih bo možno dodajati. Prav tako jim bo možno spreminjati pravice ter spreminjati osebe, ki določeni vlogi pripadajo. V nadaljevanju sledi opis vlog, ki jih bomo uporabljali, sicer pa bodo vloge določljive (ne bodo fiksne).

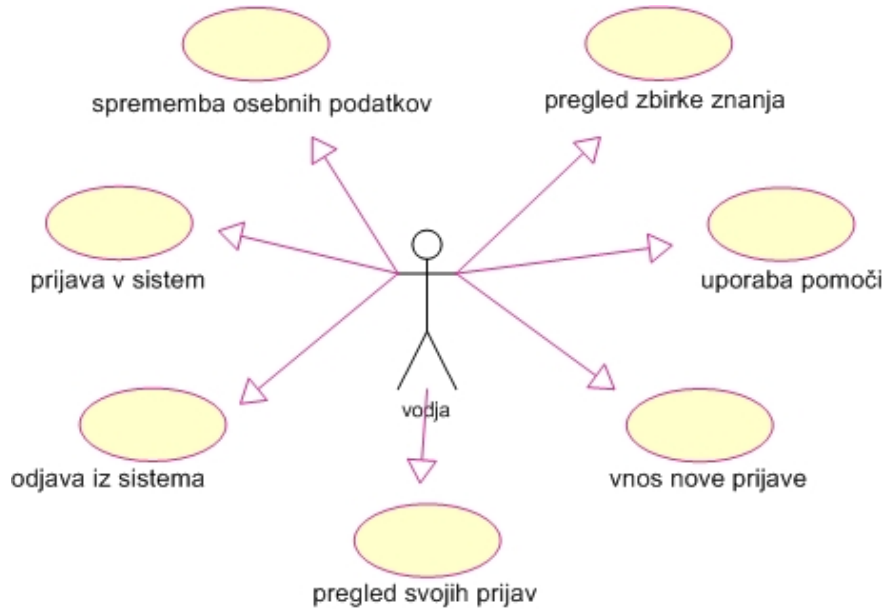
Osebe, ki bodo uporabljale aplikacijo, bodo prevzele eno izmed naslednjih vlog:

- **Uporabnik;** To vlogo bo imel vsak, ki se prijavi v aplikacijo.
- **Skrbnik;** Za vsak projekt se določi, kdo je skrbnik projekta, in potem se vse zadeve, ki se tičejo tega projekta, posredujejo najprej tej vlogi.
- **Ekspert;** Ekspert je strokovnjak za neko področje. Vsak skrbnik bo imel podatke o znanju eksperta in projektih na katerih je delal.
- **Administrator;** Glavni skrbnik sistema. Vloga je podobna vlogi skrbnik, le da lahko administrator tudi spreminja šifrante.
- **Glavni uporabnik;** To je oseba na strani podjetja naročnika, ki ima pravice prijavljanja uporabnikov podjetja in pravice pregleda vseh podatkov o podjetju v bazi helpdesk aplikacije.

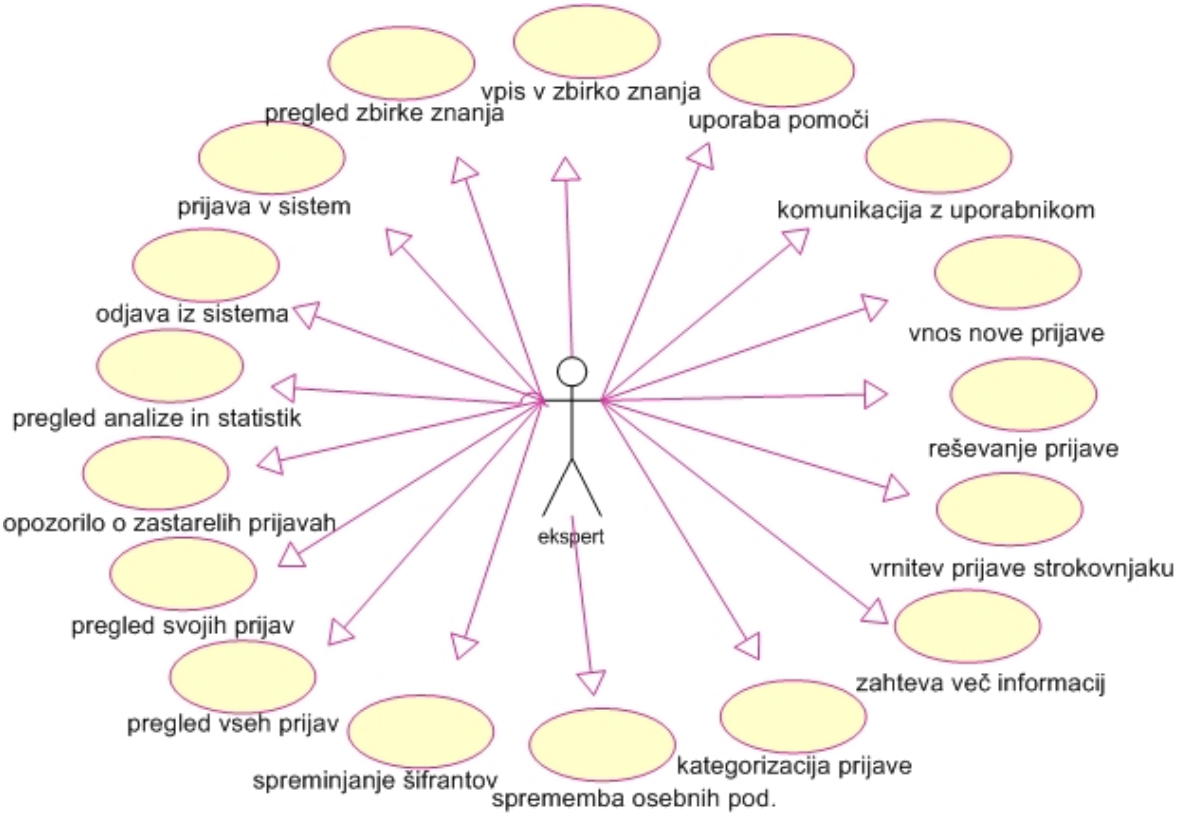
[61] obsežna baza opisov različnih težav in situacij, s katerimi so se srečavali uporabniki ter njihove rešitve. Pokriva postopke presoje kakovosti: organiziranje presoj kakovosti, beleženje odkritih napak, izdajanje korektivnih ukrepov in izdelavo poročil o kakovosti.

- **Vodja;** Nadzornik na naši strani. Vloga ima različne pravice. Vodja podjetja lahko pregleduje vse, Vodja skupine pa samo tiste zadeve, ki se tičejo projektov njegove skupine.

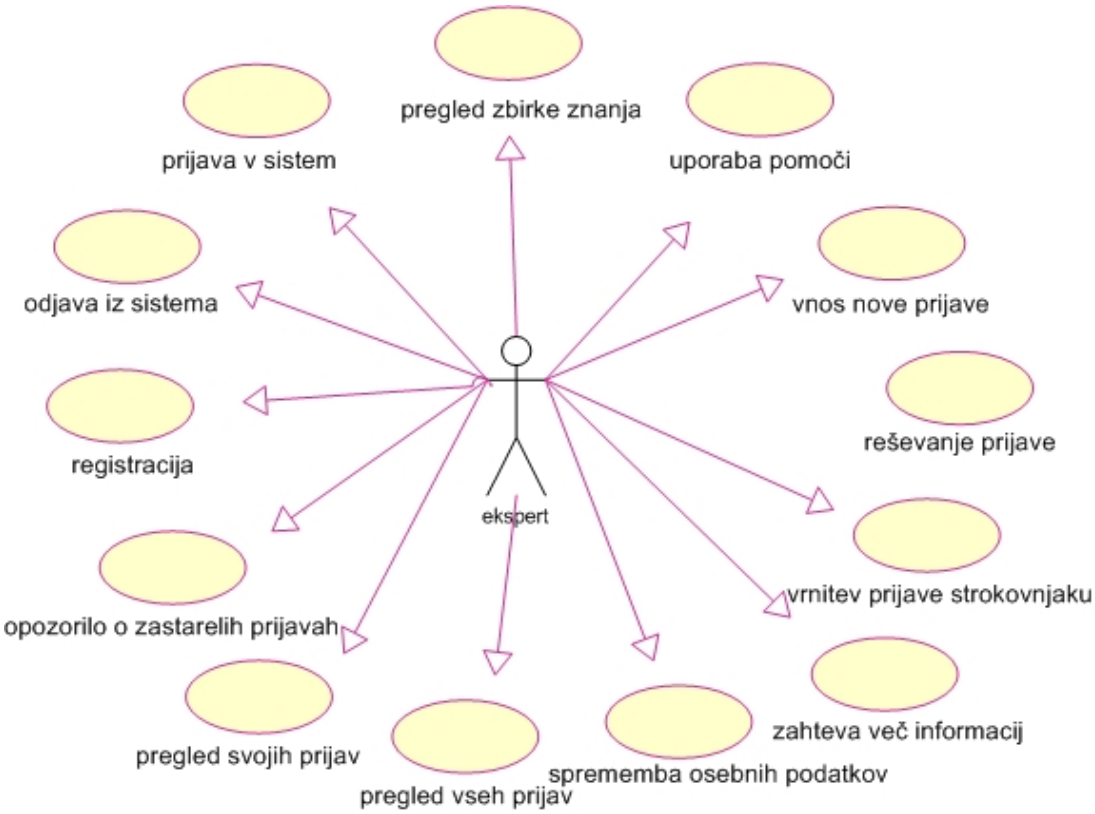
**Primeri uporabe posameznih vlog**



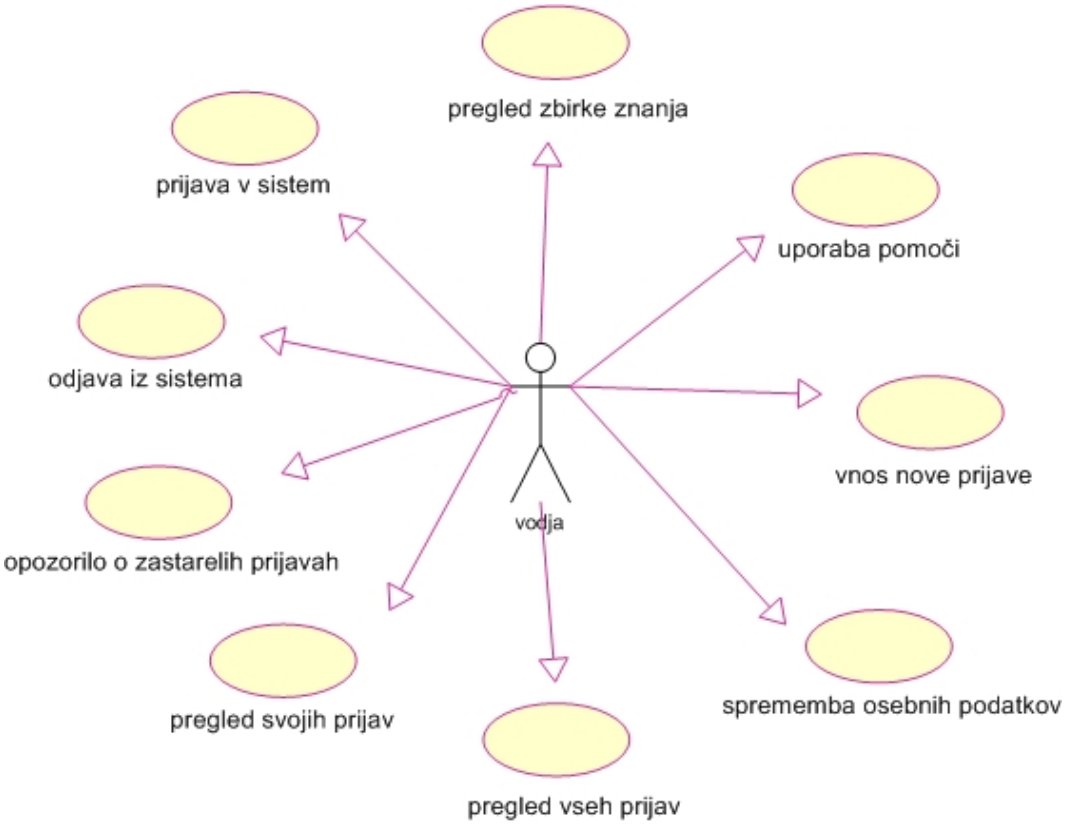

**Slika 3.1 Primer uporabe za vlogo Uporabnik**



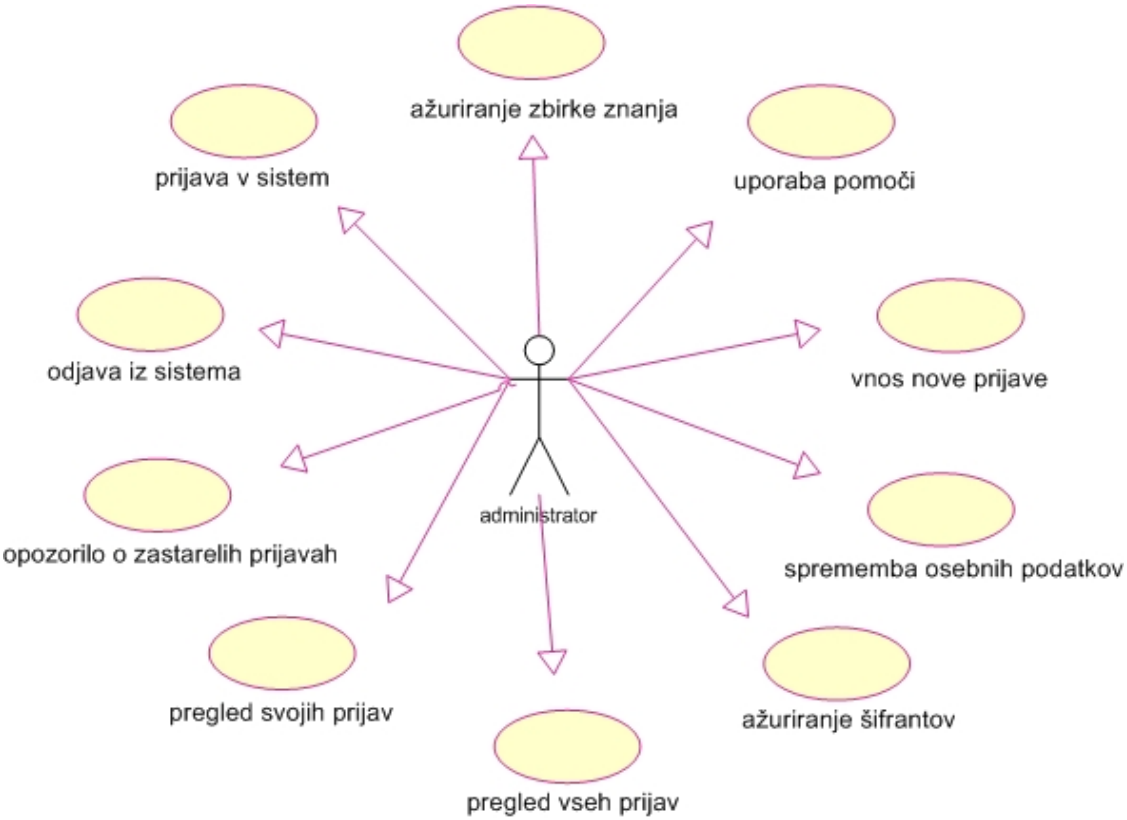

**Slika 3.2 Primer uporabe za vlogo Skrbnik**




**Slika 3.3 Primer uporabe za vlogo Ekspert**




**Slika 3.4 Primer uporabe za vlogo Vodja**




**Slika 3.5 Primer uporabe za vlogo Administrator**

## *3.2 Ozadje odločitve za uporabo Bugzille*

Pri Comlandu smo se najprej odločili za lasten razvoj helpdesk aplikacije na podlagi prej podanih specifikacij. Aplikacija naj bi se razvila v Microsoft .NET tehnologiji, kot aplikacijski strežnik bi uporabili Microsoft IIS[62], kot podatkovni strežnik pa Microsoft SQL Server. Ocenili smo, da bo za celoten razvoj aplikacije potrebnih okrog 2000 ur. Lasten razvoj aplikacije smo dejansko tudi začeli (postavili podatkovno bazo in delno začeli s programiranjem), vendar se je razvoj, zaradi preobremenjenosti kadrov, večkrat prekinil. Glede na to, da je šlo za naš interni projekt, so imeli projekti naših naročnikov prednost. Razvoj helpdesk aplikacije se je tako precej zavlekel, pravega, konkretnega napredka pa ni bilo. Bili smo že v precejšnji časovni stiski, saj smo aplikacijo nujno potrebovali za popoln prevzem vzdrževanja aplikacije e-student od Fakultete za Računalništvo in Informatiko. Zato smo sprejeli odločitev, da prekinemo z razvojem lastne helpdesk aplikacije in poiščemo drugo rešitev. Ena od možnosti je bila, da kupimo kakšno od že obstoječih helpdesk aplikacij na

[62] Internet Information Server je eden od komponent operacijskega sistema Windows in služi za komunikacijo med uporabnikom in aplikacijo

tržišču. Vendar se nam to ni zdelo primerno, saj prinaša taka rešitev kar nekaj slabosti, ki so za nas nesprejemljive:

- veliki stroški nakupa
- vsakoletno obnavljanje licence (dodatni stroški)
- neprilagojenost našim potrebam
- nemožnost lastnih nadgradenj in izpopoljevanje programske opreme (odvisnost od dobavitelja)
- nemožnost nadaljnega trženja aplikacije

### 3.2.1 Iskanje in izbiranje med odprtokodnimi rešitvami

Kot na dlani se je tako ponujala druga možnost, to je pogledati v svet odprte kode in tam najti ustrezno rešitev. Sprva smo bili glede te odločitve precej skeptični, saj se je prvič zgodilo, da se je v podjetju o odprti kodi razmišljalo kot o ustrezni rešitvi. Tako je odločitev zahtevala določen zasuk v glavah razvijalcev in vodstva, potrebno se je bilo znebiti negativnih pomislekov glede odprte kode in jo sprejeti kot možnost, ki nam ponuja nove priložnosti.
Glede na to, da nismo imeli izkušenj z uporabo prostega programja, nismo vedeli, kje je takšno programje najbolje iskati. Zato smo se poslužili kar iskalnika Google[63], kjer smo iskali po ključnih besedah *bug tracking system* in *open source*. Našli smo veliko programov, ki so oglaševali svoje rešitve sledenja prijavam napak, hroščev, predlogov in podpore uporabnikom. Iz te množice je bilo potrebno izluščiti dovolj kakovostne rešitve. Pri tem smo si pomagali s spletnimi stranmi, kjer so organizirano zbrani in predstavljeni odprtokodni projekti. Taki strani sta že prej omenjeni *sourceforge*[64] in *freshmeat*[65]. Za vsak projekt na teh straneh so podani podatki o statusu razvoja projekta (v kateri fazi razvoja je projekt), razširjenosti, oceni uporabnikov, uporabljeni tehnologiji, uporabljeni licenci, podatki o tem, na katerih operacijskih sistemih deluje itd. Za nas pomembni kriteriji so bili:

- program mora delovati v Windows operacijskem sistemu
- program mora biti v zreli (produkcijski) oziroma stabilni fazi razvoja
- program mora imeti široko množico uporabnikov, saj je to eden od pogojev za uspešnost odprtokodnega projekta

[63] http://www.google.com
[64] http://sourceforge.net
[65] http://freshmeat.net

- program mora biti narejen v spletni tehnologiji (PHP, ASP, Perl), tako da končnim uporabnikom zadostuje že spletni brskalnik za njegovo uporabo
- pomembna je bila tudi čim višja ocena uporabnikov, ki so program preizkusili

### 3.2.2 Sprejem odločitve

Na podlagi teh kriterijev so v ožji izbor prišle naslednje tri odprtokodne aplikacije:

- Anthill[66]
- Mantis[67]
- Bugzilla[68]

Po preizkusu vseh treh demo (testnih) različic, ki so postavljene na spletu[69], nas je najbolj prepričala Bugzilla s svojo superirorno funkcionalnostjo. Izpolnjevala je namreč vse naše zahteve, ki smo jih podali v specifikaciji helpdesk aplikacije, medtem ko sta bili aplikaciji Anthill in Mantis na tem področju nekoliko pomanjkljivi (predvsem glede možnosti administracije skupin uporabnikov in njihovih pravic). V prid Bugzille je govorila tudi licenca, pod katero se ta aplikacija razširja. Bugzilla namreč uporablja MPL licenco, ki je za razliko od GPL licence, pod katero sta aplikaciji Anthill in Mantis, precej manj omejevalna. Med drugim omogoča, da nadgradnje oz. razširitve programa lahko v primeru nadaljnega razširjanja spremenjene binarne (izvršljive) različice programa, zadržimo zase in jih ne razširjamo naprej. Zelo pomemben faktor je predstavljal spisek podjetij ali organizacij, ki uporabljajo določen program. Pri tem je bila Bugzilla najbolj impresivna, namreč do danes jo uporablja kar 348 podjetij in organizacij. Med njimi so najbolj znane NASA, IBM, Red Hat, Mozilla, Apache projekt in drugi.

Aplikaciji Anthill in Mantis sta bili v prednosti predvsem pri izgledu uporabniškega vmesnika in težavnosti namestitve ter konfiguracije. Pri tem je v pozitivnem smislu Mantis najbolj izstopal, medtem ko smo pri Bugzilli spoznali, da je njen uporabniški vmesnik daleč od sodobnih trendov, zelo nepregleden in preveč kompleksen za navadnega uporabnika. Prav tako je njen postopek namestitve dokaj zapleten in zahteva dobro poznavanje Apache strežnika ter Perl modulov. Vendar je po drugi strani dobro dokumentiran, kar delno zmanjšuje njegovo kompleksnost. Kljub omenjenima pomanjkljivostima Bugzille smo bili

[66] http://anthill.vmlinuz.ca
[67] http://www.mantisbt.org
[68] http://www.bugzilla.org
[69] Anthill demo se nahaja na: http://anthill.vmlinuz.ca/demo/, Mantis demo se nahaja na: http://www.mantisbt.org/demo.php, Bugzilla demo pa je na naslovu: http://landfill.bugzilla.org/

mnenja, da njena funkcionalnost in razširjenost uporabe prevesita tehtnico v njeno korist. Tako smo se dokončno odločili za uporabo Bugzille kot naše helpdesk aplikacije.

## *3.3 Bugzilla*

### 3.3.1 Kaj je Bugzilla

Bugzilla je sistem za sledenje hroščev oziroma napak v programski opremi. Poleg tega omogoča posamezniku ali skupini razvijalcev učinkovito slediti predlogom za razširitve, problemom in drugim zadevam, povezanim s projekti razvoja programske opreme. Prvotno verzijo Bugzille je razvil Terry Wiesmann v programskem jeziku TCL[70] z namenom nadomestiti okrnjen sistem za sledenje hroščev, ki so ga interno uporabljali pri Netscape Communications. Terry jo je kasneje pretvoril v programski jezik Perl, v katerem je ostala vse do danes. Večina proizvajalcev komercialnih sistemov za sledenje napakam v programih zahteva precejšnje zneske za licenco, zato je Bugzilla kmalu postala zelo razširjena po celem svetu.

### 3.3.2 Kaj Bugzilla ponuja

V osnovi je Bugzilla namenjena:

- sledenju hroščem (napakam) in spremembam v programski kodi
- komunikaciji med razvijalci
- posredovanju in pregledovanju popravkov, nadgradenj (ang. patch)
- nadzoru zagotavljanja kakovosti (ang. quality assurance)

Bugzilla pripomore k obvladovanju procesa razvoja programske opreme. Uspešni projekti so običajno rezultat uspešne organizacije in komunikacije, Bugzilla pa je orodje, ki omogoča učinkovito organiziranje in komuniciranje skupine pri projektih. Odprt sistem za sledenje napak omogoča proizvajalcem ostati v stiku s svojimi strankami s pomočjo učinkovite komunikacije. Bugzilla zmanjšuje čas nedelovanja sistema (ang. downtime), poveča produktivnost in zadovoljstvo uporabnikov ter izboljša komunikacijo. Prav tako pomaga zmanjšati stroške z zagotavljanjem IT podprte odgovornosti (ang. accountability), zbirk znanja (ang. knowledge base) za telefonsko podporo in kontrole nad nenavadnimi sistemskimi

[70] TCL je odprtokodni skriptni jezik, ki deluje na Windows, Macintosh in Unix osnovanih sistemih.

ali programskimi problemi. Zelo je prilagodljiva različnim situacijam oziroma okolju, saj jo je možno uporabiti na področjih kot so:

- sistemska administracija
- upravljanje z namestitvami[71] (ang. deployment management)
- sledenje problemom pri načrtovanju in razvoju čipovja (pred in po izdelavi)
- slednje hroščem (napakam) programske in strojne opreme

### 3.3.3 Funkcionalnost Bugzile

Pri predstavitvi funkcionalnosti se ne bom spuščal v podrobnosti, temveč bom le okvirno predstavil glavne funkcije, ki jih Bugzilla omogoča.

**Administracija**

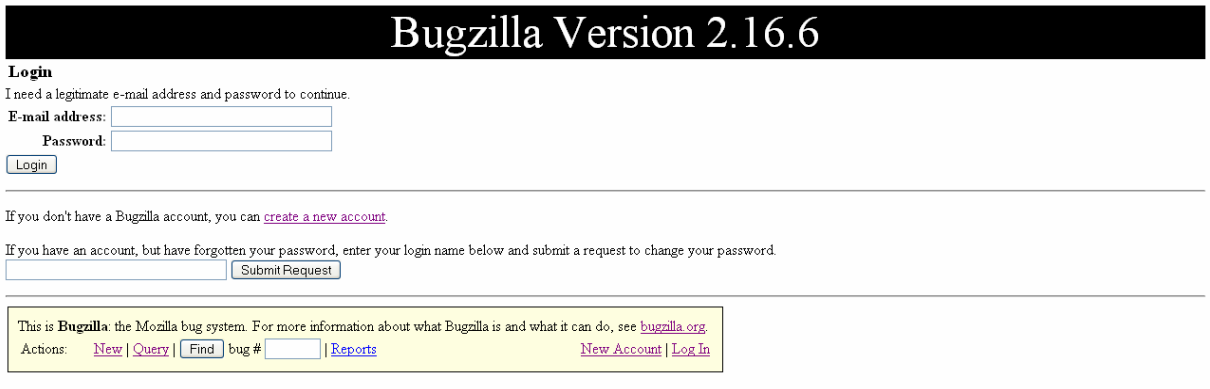
Administracija v Bugzilli vključuje:

- **Administracija uporabnikov**; Možno je kreiranje in brisanje »super« uporabnikov (administratorjev), kreiranje, brisanje in spreminjanje lastnosti navadnih uporabnikov.
- **Administracija produktov, komponent, verzij in mejnikov**; *Produkti* so najbolj obsežna kategorija in predstavljajo produkte (aplikacije), za katere se namerava preko Bugzille nuditi uporabniška pomoč. *Komponente* so podkategorije produktov. Vsak produkt je lahko sestavljen iz več komponent (moduli oziroma vsebinsko ločeni deli produkta). *Verzije* predstavljajo različne izdaje (revizije) produkta. *Mejniki* so cilji, ki se jih želi doseči s popravki napak (hroščev) v produktu. Produkte, komponente, verzije in mejnike je možno dodajati, odstranjevati ter spreminjati njihove lastnosti.
- **Administracija skupin in pravic**; Administratorji lahko s skupinami izolirajo napake in produkte tako, da lahko do njih dostopa samo določena skupina uporabnikov. Bugzilla pozna dva tipa skupin: *skupine na osnovi produktov* (skupine, ki so vezane na produkte in se ustvarijo ob kreiranju produkta), *splošne skupine* (nimajo povezave s produkti). Administrator lahko skupine kreira, odstranja in spreminja. Lahko dodaja in odstranja uporabnike skupin ter določa pravice skupine (in njenih uporabnikov).
- **Glasovanje**; Uporabnikom v Bugzilli je možno dodeliti določeno število glasov, ki jih lahko oddajo posameznim napakam (en glas eni napaki) in s tem razvijalcem omogočijo, da lažje ocenijo potrebo uporabnikov po določenem popravku ali razširitvi produkta. Glasovanje je možno vklopiti ali izklopiti.

[71] namestitev vključuje instalacijo programske opreme na ciljne računalnike ter vso potrebno konfiguracijo (nastavitev parametrov) sistema ali programske opreme.

### Kreiranje uporabniškega računa in prijava v sistem

Vsakdo, ki želi prijaviti zadevo (oddati zahtevo za spremembe v produktu, spremembo funkcionalnosti, za razširitev produkta ali prijavo napake) oziroma na splošno kakorkoli uporabljati funkcije Bugzille, mora biti prijavljen v sistem, torej mora imeti v Bugzilli kreiran uporabniški račun. To lahko stori uporabnik sam, ali pa administrator. Slika 3.6 prikazuje obrazec za prijavo v Bugzillo.



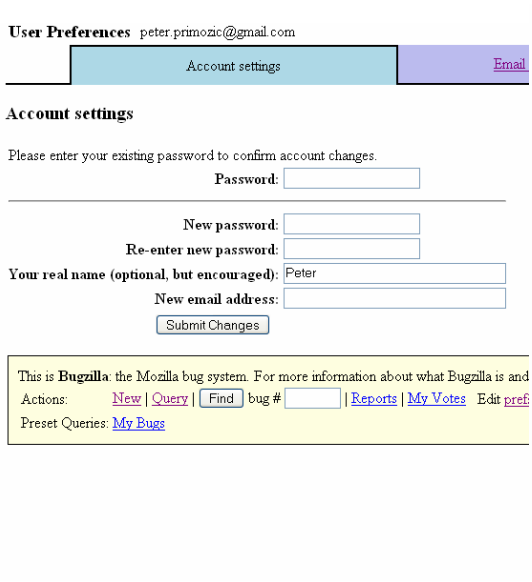

**Slika 3.6 Prijava v sistem**

### Uporabniške nastavitve

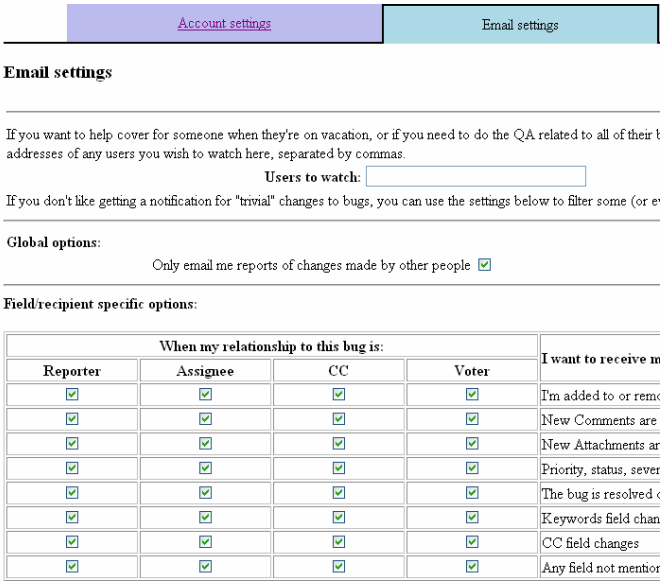
Uporabniki lahko Bugzillo v določeni meri prilagodijo svojim potrebam. Možno je konfigurirati:

- Nastavitve računa (elektronski naslov za prejemanje pošte, geslo za vstop v sistem itd.)
- Nastavitve e-pošte (nastavlja se ob katerih akcijah v Bugzilli, bo obvestilo o tem poslano uporabniku. Akcije so npr. sprejem prijave zadeve, sprememba statusa prijave itd.)
- Dovoljenja (tukaj gre samo za informativni prikaz pravic uporabnika in včlanjenosti v skupine)

Slika 3.7 prikazuje obrazec za spreminjanje nastavitev uporabnišjega računa, slika 3.8 pa obrazec za spreminjanje nastavitev elektronske pošte.




**Slika 3.7 Nastavitve računa**

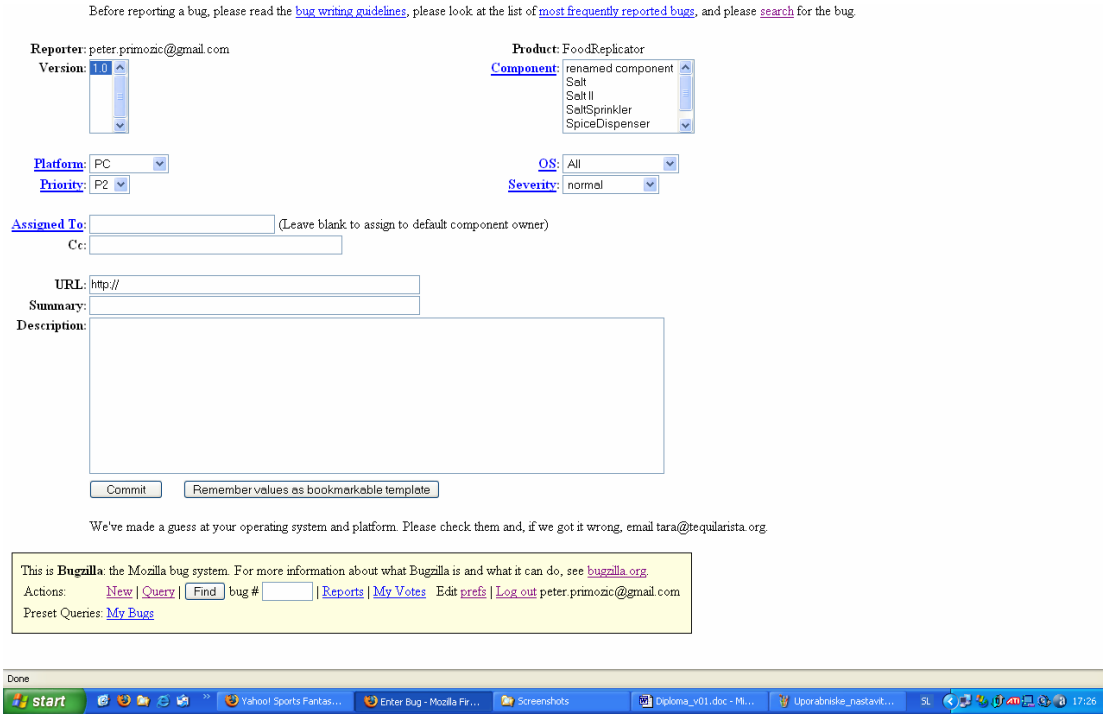



**Slika 3.8 Nastavitve elektronske pošte**

### Prijava in reševanje zadeve

Uporabnik mora biti za prijavo zadeve prijavljen v sistem. Prijava zadeve poteka tako, da se najprej izbere produkt, nato pa se izpolnijo polja (atributi) prijave. Vsaka prijava se avtomatsko dodeli tistemu, ki je zadolžen za reševanje zadev določenega produkta oziroma komponente. Po prijavi dobi le-ta status *Nova* ali *Nepotrjena*. Ko razvijalec dobi obvestilo o prijavi, lahko postavi njeno stanje na *Dodeljena* (če jo dodeli v reševanje bodisi sebi bodisi dugemu razvijalcu). Ko je prijava rešena, pa postavi njeno stanje na *Rešeno*. Možni sklepi rešitve so: *Popravljeno, Nerešljivo, V naslednji verziji* itd. Rešeno prijavo je zaradi zagotavljanja kakovosti potrebno preveriti (ali je zares uspešno rešena). Takrat dobi status *Preverjena*. Nato jo je možno zaključiti (dobi status *Zaključena*). Če prijava ni uspešno rešena, jo je možno ponovno odpreti (dobi status *Ponovno odprta*). S tem se zopet vrne v stanje, ko jo mora določen posameznik sprejeti v reševanje.

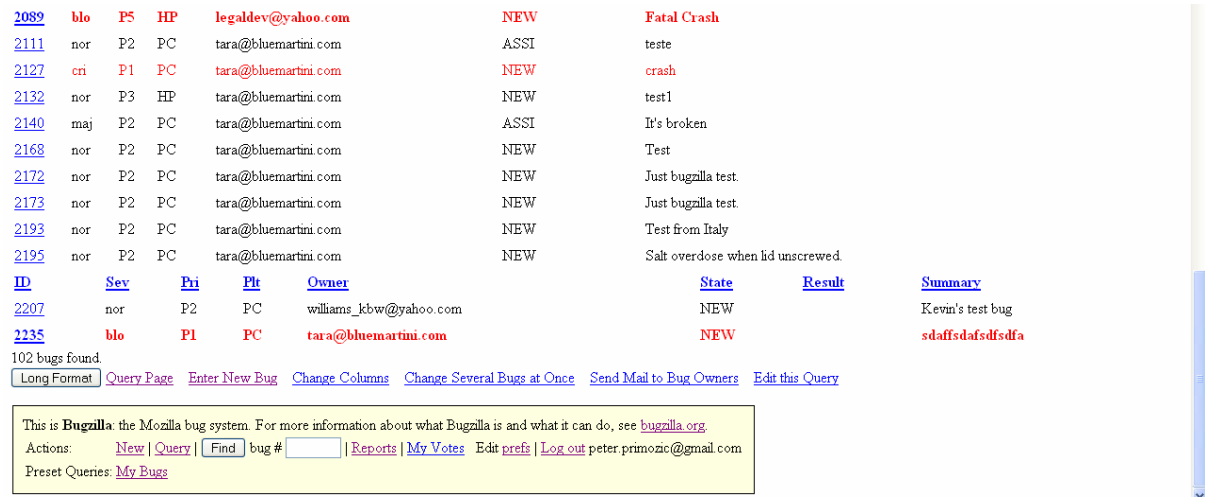
Vse akcije na prijavi se beležijo. Omogčen je vpogled v zgodovino dogodkov prijave, s tem je možno v vsakem trenutku ugotoviti, kaj se je z njo dogajalo. Na sliki 3.9 je prikazan obrazec za podajo prijave zadeve.



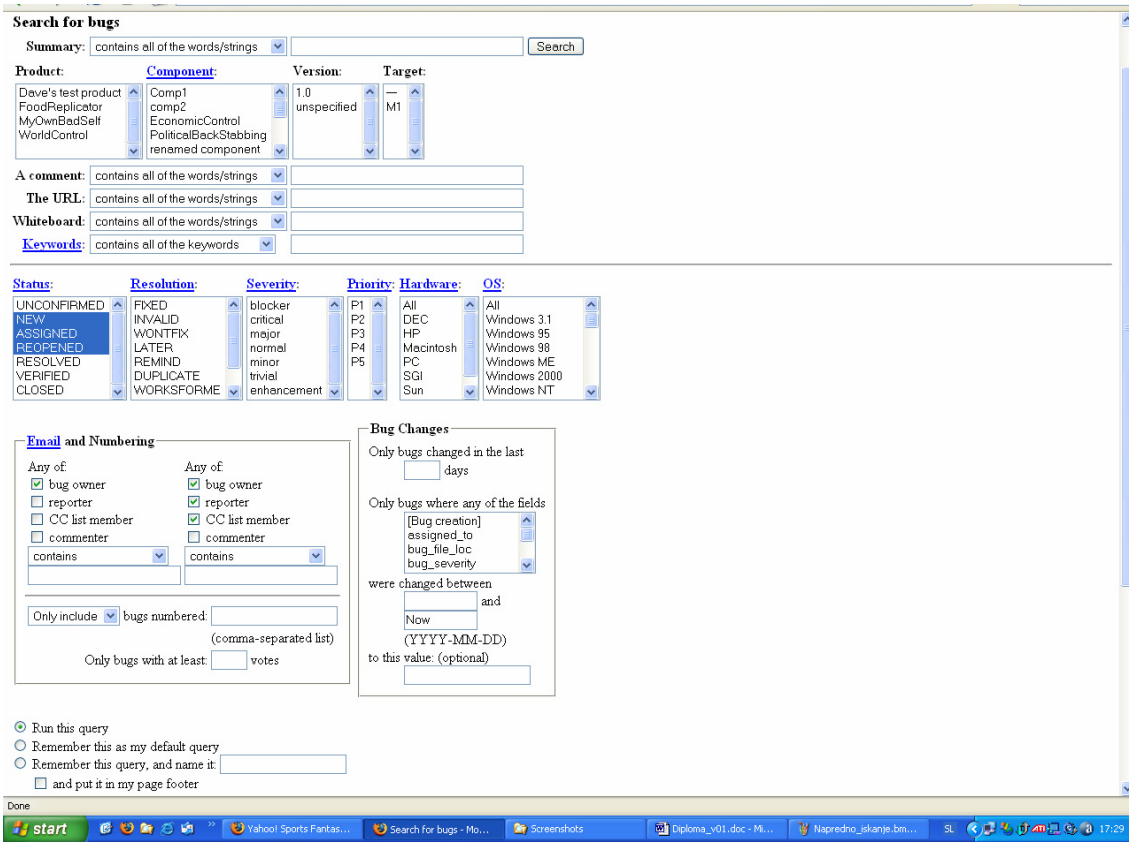

**Slika 3.9 Prijava zadeve**

### Iskanje in pregled prijav

Bugzilla vsebuje zelo zmogljiv iskalnik, ki omogoča iskanje po vrsti različnih kriterijev (po vseh atributih prijave). Iskalne poizvedbe je možno shraniti (avtomatsko se naredi povezava na osnovni strani uporabnika) in jih tako naslednjič hitro priklicati z enim samim klikom na povezavo. Rezultat poizvedbe je seznam prijav, ki ustrezajo iskalnemu pogoju. Ta seznam je možno oblikovati (dodajati ali odstranjevati stolpce seznama, določiti format izpisa: CSV ali tako imenovan Long format). Slika 3.10 prikazuje seznam prijav kot rezultat poizvedbe, slika 3.11 pa obrazec za iskanje prijav.



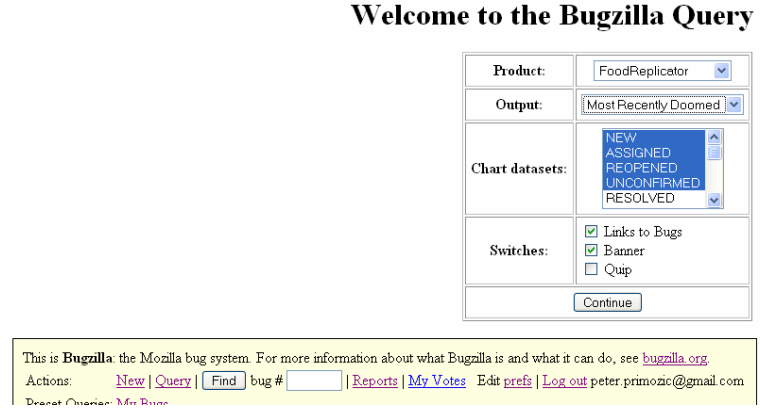

**Slika 3.10 Pregledovanje prijav**




**Slika 3.11 Iskanje prijav**

## Poročila in grafi

Bugzilla omogoča izdelavo statističnih poročil in grafov. Poročilo je prikaz trenutnega stanja baze prijav. Možno ga je izdelati v obliki HTML[72] tabele ali grafičnega prikaza (stolpični grafi, graf v obliki torte itd.). Graf je prikaz stanja baze skozi čas. Slika 3.12 prikazuje obrazec za izdelavo poročila.




**Slika 3.12 Izdelava poročila**

[72] kratica za Hypertext Markup Language; jezik, ki se uporablja za izdelavo internet strani oziroma za predstavitev informacij na medmrežju

## *3.4 Prilagajanje Bugzille*

Po odločitvi za uporabo Bugzille in njenem temeljitejšem preizkusu smo prišli do naslednjih ugotovitev:

- potrebna je lokalizacija; to je prevod vse vsebine, ki jo Bugzilla prikazuje, v slovenski jezik, pretvorba formata datumskih zapisov v našo obliko (dd.mm.llll[73]).
- uporabniški vmesnik je potreben popolne prenove; potrebno ga je prilagoditi navadnim uporabnikom, saj je v osnovi namenjen razvijalcem (programerjem).
- določene podrobnosti (funkcionalnosti) je potrebno odstraniti, ker so za navadne uporabnike nepotrebne.
- potrebno je vgraditi oziroma omogočiti komunikacijo Bugzille z drugimi (zunanjimi) aplikacijami, ki jih uporabljajo naši uporabniki. Predvsem se mora omogočiti integracijo Bugzille in sistema e-studenta[74], tako da bi lahko njegovi (e-studentovi) uporabniki, kar preko njega pošiljali prijave, predloge in zahteve, ki bi se nato shranjevale v Bugzilli.

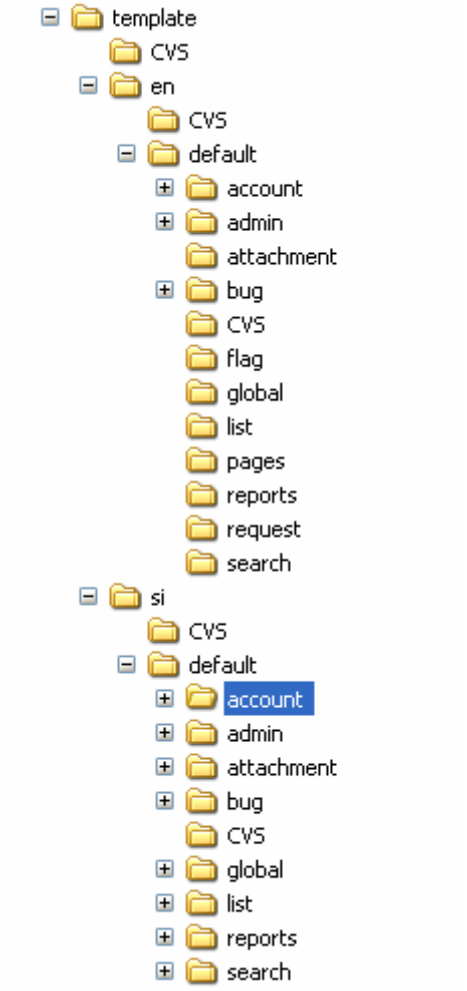
### 3.4.1 Lokalizacija Bugzille

Kot sem že omenil, je Bugzilla napisana v skriptnem jeziku Perl. Pri nas ni bilo nikogar, ki bi imel s programiranjem v tem jeziku kakršnekoli izkušnje. To je pomenilo, da je bil potreben čas, da se tega jezika naučimo, spoznamo njegovo sintakso in morebitne posebnosti. Da bi vodstvo podjetja dokončno prepričali o smotrnosti uporabe Bugzille, smo potrebovali hitre rezultate, ki bi to potrdili. Tako smo se odločili, da bo prvi korak pri prilagoditvi Bugzille njeno poslovenjenje. Pri tem nam je šlo na roko dejstvo, da ima Bugzilla v verziji 2.16, ki smo jo mi uporabili, že ločeno prezentacijsko[75] in procesno[76] logiko. Prav tako so posamezni deli prezentacijske logike (na primer: začetna stran, glava, noga spletne strani) med seboj ločeni in shranjeni v posebnih datotekah, tako imenovanih predlogah (ang. template). Taka struktura programa prinaša lažje in hitrejše spreminjanje izgleda uporabniškega vmesnika. Tako se za potrebe spreminjanja videza programa ni potrebno spuščati v modificiranje kompleksne

[73] standardna oblika zapisa datuma, kjer je d=dan, m=mesec, l=leto. Na primer 21.11.2004.
[74] študijski informacijski sistem, ki uporabnikom omogoča oddaljen dostop do podatkov in storitev, pomembnih za izvajanje študijskega procesa. Eden od spletnih naslovov, kjer se e-student nahaja: http://estudent.fri.uni-lj.si
[75] logika, ki skrbi za vizualni prikaz (prezentacijo) vsebine.
[76] logika, ki skrbi za pravilno izvajanje posameznih procesov (postopkov) programa, ki jih sprožajo uporabniki s svojimi akcijami v njem. Poskrbi za pravilno prikazovanje vsebine.

logike Perl-ovih datotek. Perl datoteke so namreč tiste, ki v Bugzilli predstavljajo procesno logiko in glede na naše nepoznavanje Perl-a smo se jim na začetku želeli izogniti.

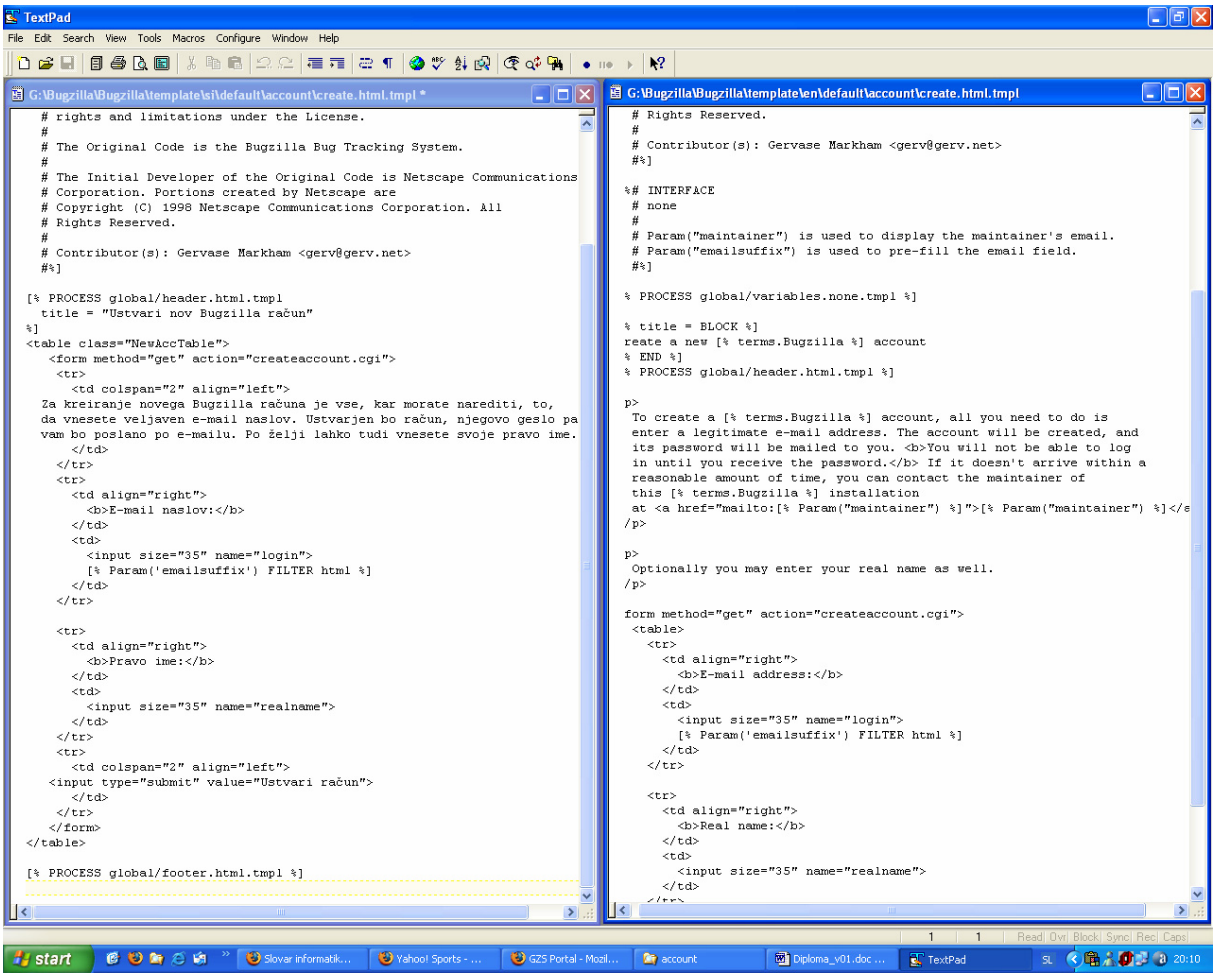
Struktura direktorijev datotek predlog v Bugzilli je takšna, da je na najvišjem nivoju direktorij *template*, znotraj katerega se nahajajo direktoriji za vsako posamezno lokalizacijo. V verziji Bugzille 2.16 znotraj *template* direktorija obstaja samo lokalizacija za angleški jezik, ki je tudi privzeta lokalizacija. Naš prvi korak poslovenjenja Bugzille je bil, da smo prepisali celotno vsebino direktorija *en*, znotraj katerega se nahajajo datoteke predlóg angleške lokalizacije in ustvarili nov direktorij z nazivom *si*, kamor smo prilepili kopije datoteke iz direktorija *en*. Tako smo dobili znotraj *template* direktorija strukturo, ki jo prikazuje spodnja slika.




**Slika 3.13 Struktura template direktorija**

Postopek poslovenjenja smo nadaljevali tako, da smo z uporabo Bugzille na spletu, to je s sprehajanjem po njenih obrazcih, ugotavljali, katera so tista besedila, ki jih moramo prevesti v slovenski jezik. Ta besedila smo nato poiskali v ustreznih predlogah in jih tudi primerno prevedli. Bugzilla ima poseben parameter, s katerim povemo, katera lokalizacija se uporablja.

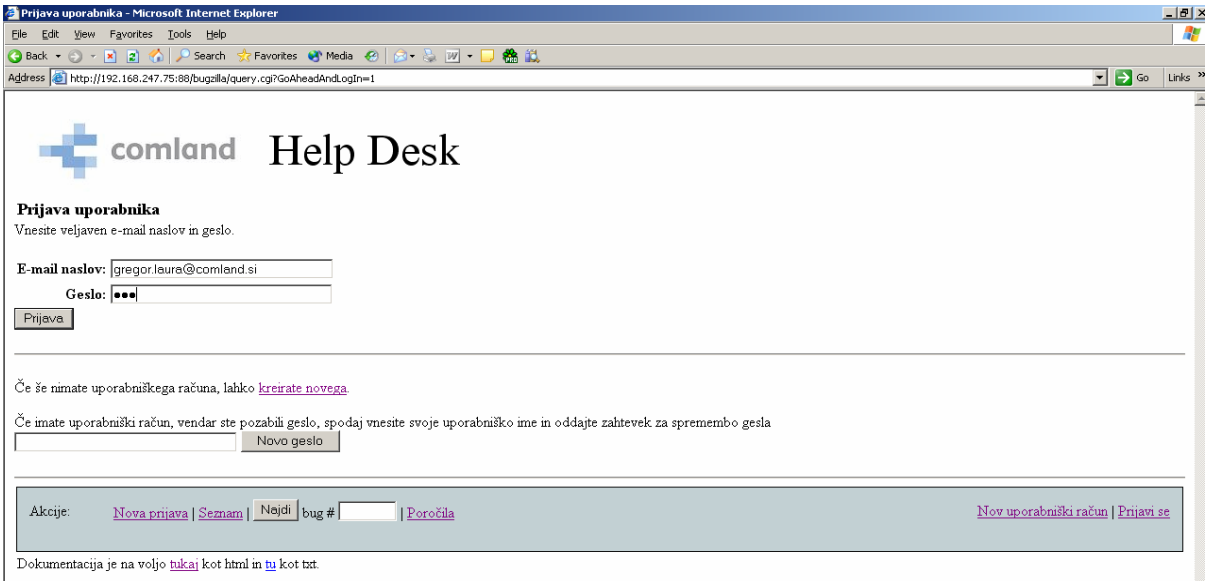
Privzeta vrednost tega parametra je bila seveda *en*. S tem, ko smo njegovo vrednost nastavili na *si*, smo naredili preklop in Bugzilla je procesirala slovenske predloge. Spodnja slika prikazuje primer angleške in slovenske predloge za kreiranje uporabniškega računa.



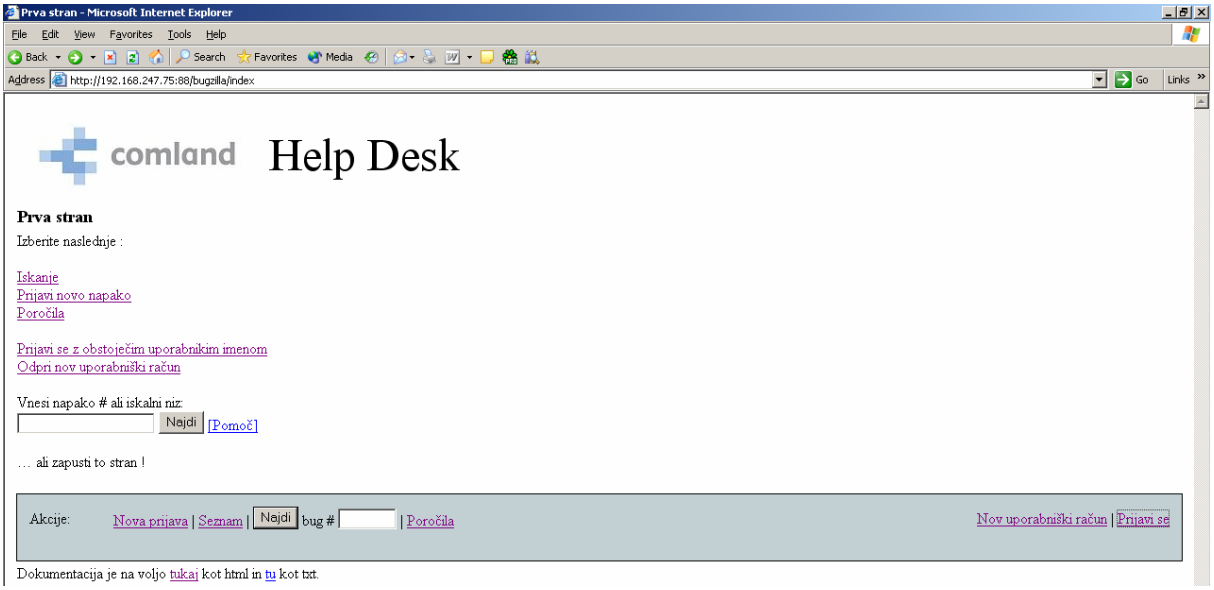

**Slika 3.14 Primejava predlog (v levem oknu slovenska, v desnem angleška)**

Postopek smo v več iteracijah ponovili. Vsakokrat, ko smo naleteli na kakšno neprevedeno besedilo, smo ga poiskali v ustrezni predlogi in ga prevedli. Rezultati prevajanja so bili hitro vidni, kar nas je vse razveselilo in še bolj prepričalo v pravilnost odločitve za uporabo Bugzille. Zavedli smo se, da Bugzilla v sebi nosi zares velik potencial in da je njena popolna vizualna preobrazba samo stvar časa. Na slikah, ki sledijo, je mogoče videti izgled naše Bugzille po njenem poslovenjenju. Omenil bi še, da je na teh slikah mogoče tudi že videti pasico[77] (ang. banner) s simbolom našega podjetja, s katero smo zamenjali osnovno pasico.

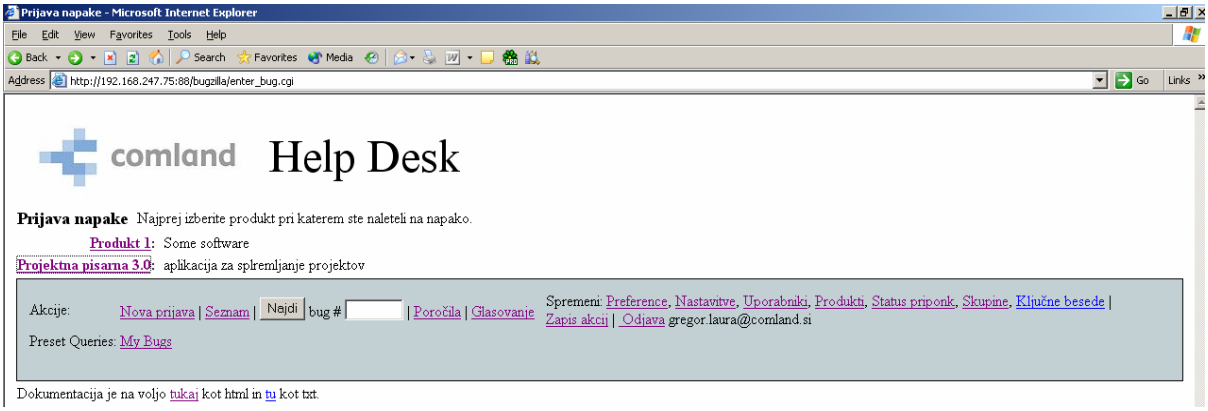
[77] ozek trak, navadno čez celotno okno na spletni strani, ki označuje ime,identiteto strani,oglašuje izdelek ali drugo spletno stran




**Slika 3.15 Obrazec za prijavo v Bugzillo**



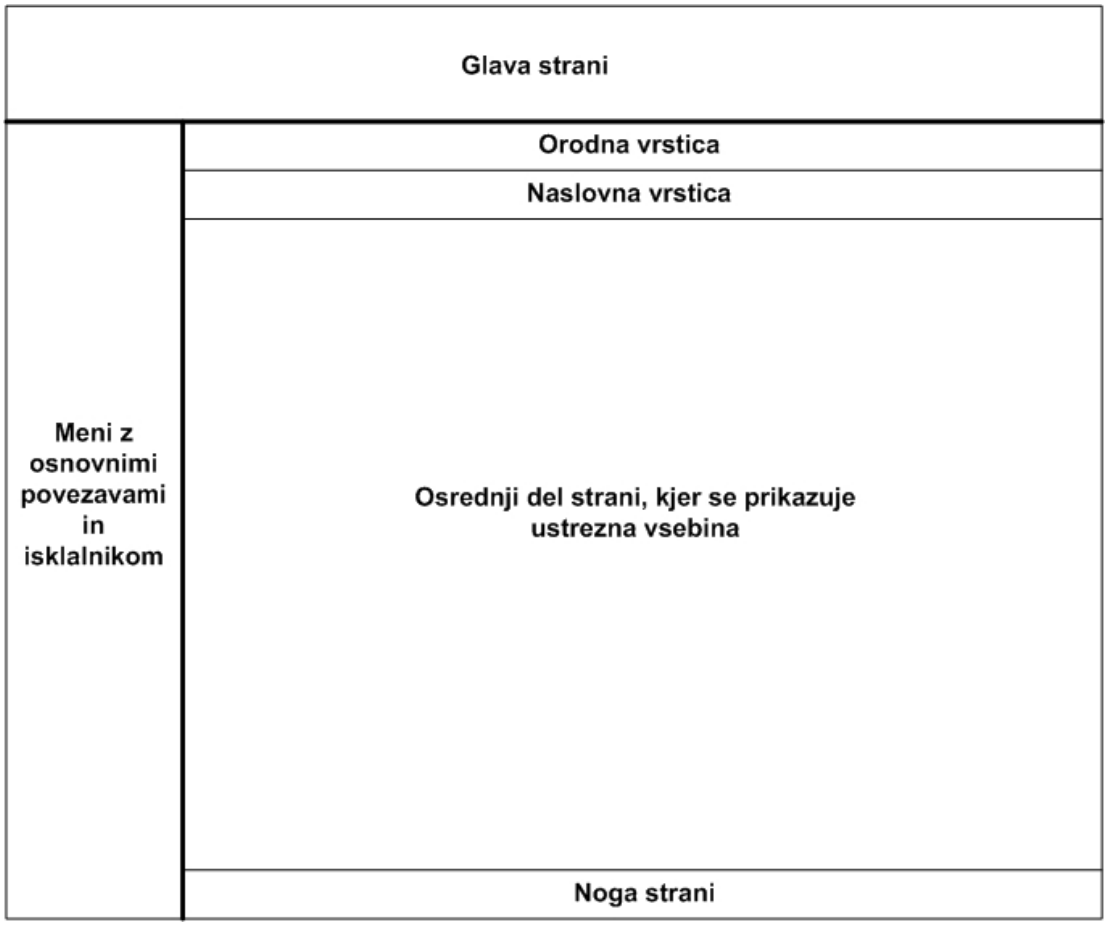

**Slika 3.16 Prva stran po prijavi v Bugzillo**




**Slika 3.17 Izbira produkta pri prijavi zadeve**

### 3.4.2 Preobrazba uporabniškega vmesnika

Da bi Bugzillo naši uporabniki lahko čimprej začeli uporabljati, je bilo nujno potrebno narediti popolno preoblikovanje njenega uporabniškega vmesnika. V ta namen smo najprej izdelali CSS[78], ki naj bi zagotovil enoten izgled vseh spletnih strani Bugzille. Nato pa smo se lotili spreminjanja videza uporabniškega vmesnika. Tudi v tem primeru je za izpolnitev našega clija zadostovalo, da smo modificirali samo predloge, ne pa tudi Perl datoteke.
Prvi korak pri preoblikovanju vmesnika je bila razdelitev spletne strani na več delov, v skladu s sodobnimi oblikovalskimi trendi, to je s pomočjo tabel (element html jezika). Na sliki 3.18 je grafično prikazana razdelitev strani.




**Slika 3.18 Razdelitev spletne strani Bugzille**

Glava strani je tabela z eno vrstico in eno celico[79]. Zapolnili smo jo s simbolom našega podjetja in nazivom aplikacije. Levi in osrednji del strani je predstavljen z drugo tabelo, ki

[78] Cascading style sheet je datoteka, v kateri se določi izgled (velikost, barva, ozadje itd.) posameznih elementov, ki sestavljajo spletno stran (glava, tabela, celice tabele, gumb, tekstovno polje, besedila).
[79] element html jezika, ki ima oznako <td> - predstavlja stolpec znotraj vrstice tabele. Vrstica ima oznako <tr>.

ima eno vrstico in dve celici. V levi del strani smo vstavili uporabniški meni s povezavami na shranjene poizvedbe za prikaz seznama prijav in personalizacijske oziroma administratorske strani Bugzille. Osrednji del je razdeljen na štiri dele. Prvi predstavlja orodno vrstico s povezavama na novo prijavo in pomoč, drugi naslovno vrstico, tretji del predstavlja prostor, kjer se prikazujejo obrazci in poročila Bugzille, četrti del pa nogo strani.

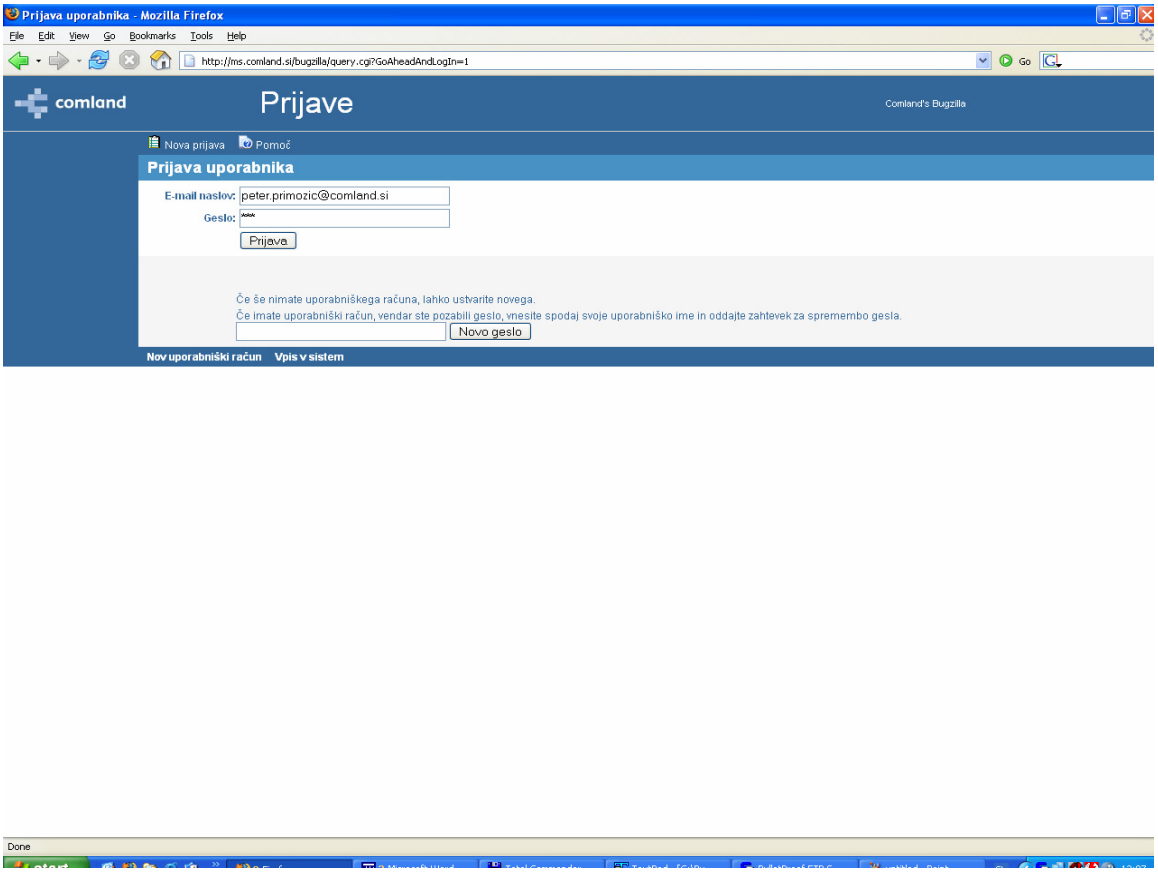
Takšno razdelitev strani smo dosegli s spreminjanjem in dodajanjem kode znotraj predloge *header*, ki je sestavni del vsake spletne strani v Bugzilli. V predlogo *header* smo vključili tudi referenco na naš CSS, s čimer smo omogočili, da se lahko v ostalih predlogah pri oblikovanju posameznih elementov spletne strani, sklicujemo na deklarirane sloge znotraj CSS-ja. Predloga *header* je namreč tista, ki se kot prva izvrši znotraj vseh drugih predlog, saj vsebuje temeljne html elemente vsake spletne strani, kot so oznaka, da gre za html stran (<html>) ter glava in telo spletne strani (html oznaka <*head*> oziroma <*body*>).

V ostalih predlogah smo predvsem spreminjali:

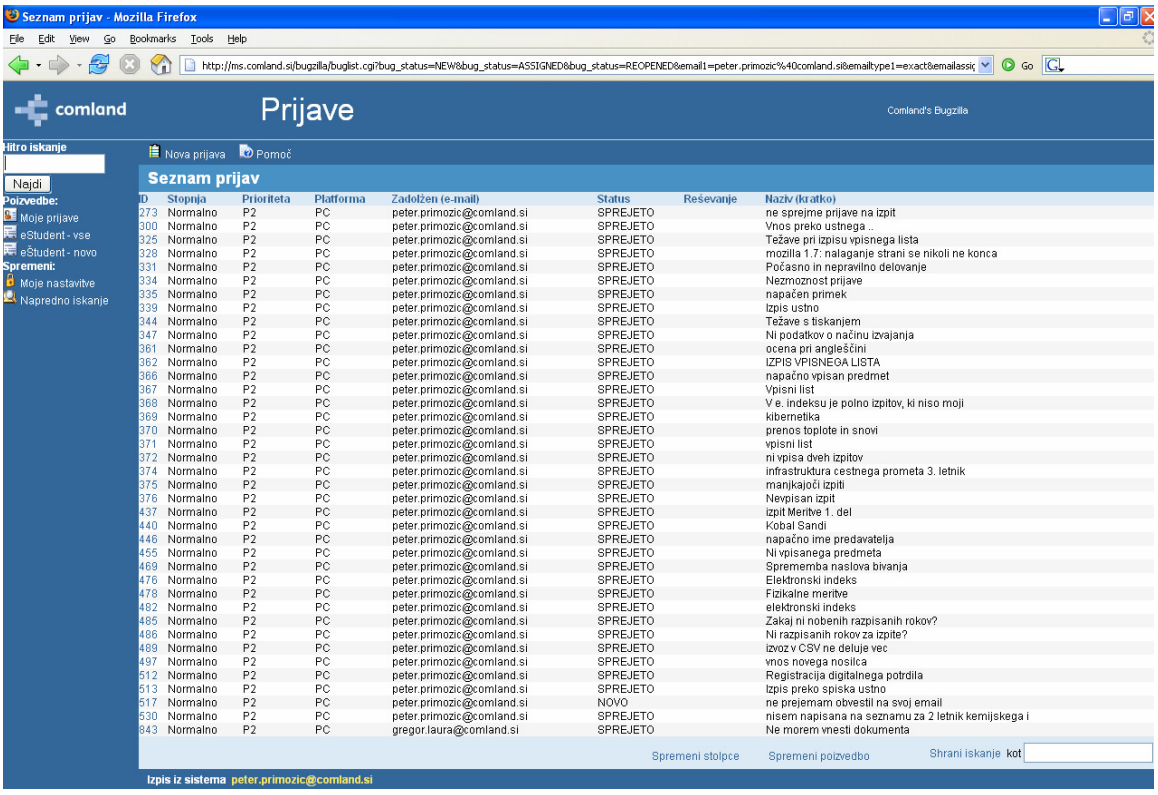
- izgled posameznih polj na obrazcih
- postavitev polj na obrazcih
- izgled poročil (npr. seznam odprtih prijav)
- povezavam smo dodali grafične ikone za lažje razločevanje in večjo prepoznavnost

Obstoječim predlogam Bugzille smo dodali tudi nekatere svoje. Tako smo med drugim na primer dodali predlogi *Loginout* ter *Toolbar1*. Prva se uporablja za nogo strani in vključuje povezave na vpis in izpis iz sistema, druga se uporablja za orodno vrstico in vključuje povezave na prijavo nove zadeve ter uporabniško pomoč (pogosto zastavljena vprašanja - FAQ).

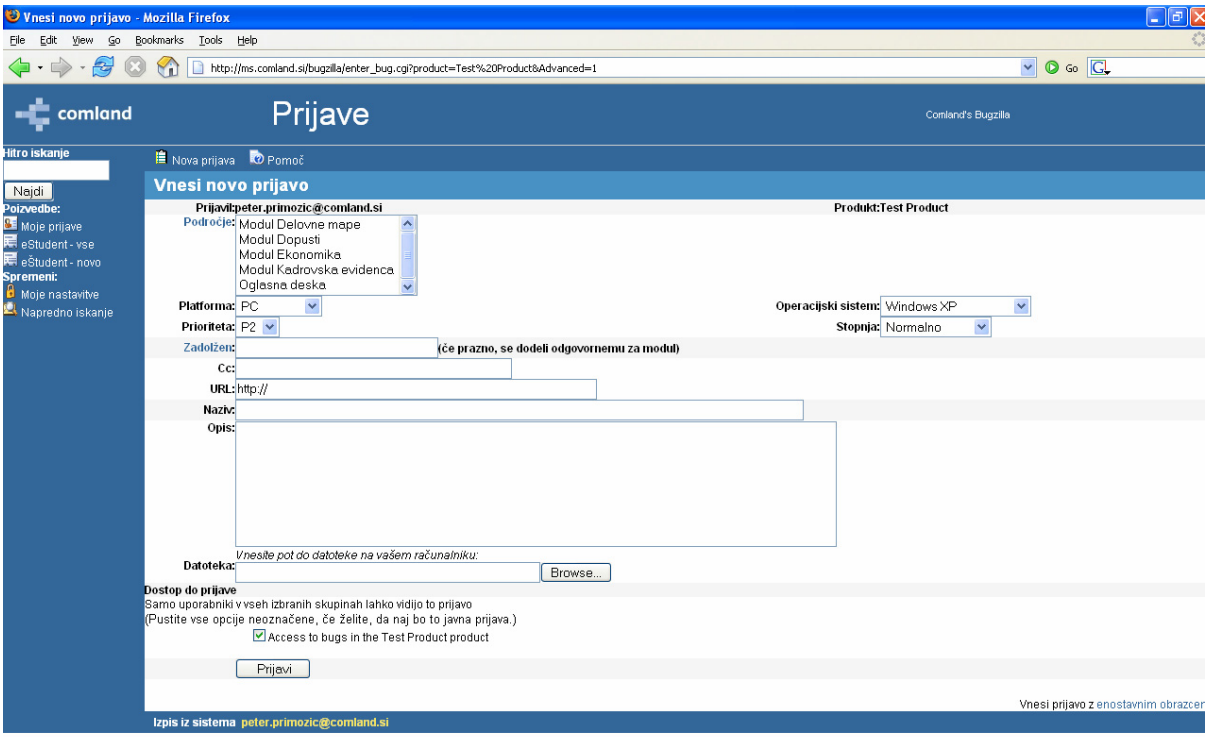
V nadaljevanju sledi nekaj slik, ki prikazujejo končen rezultat naše predelave uporabniškega vmesnika Bugzille. V primerjavi s prvotnim izgledom Bugzille, ki sem ga okvirno predstavil v poglavju 3.3.3, je opazna precejšnja razlika. To samo potrduje dejstvo, da je Bugzilla izredno prilagodljiva. Vsakdo lahko spremeni njen izgled po svoji meri.



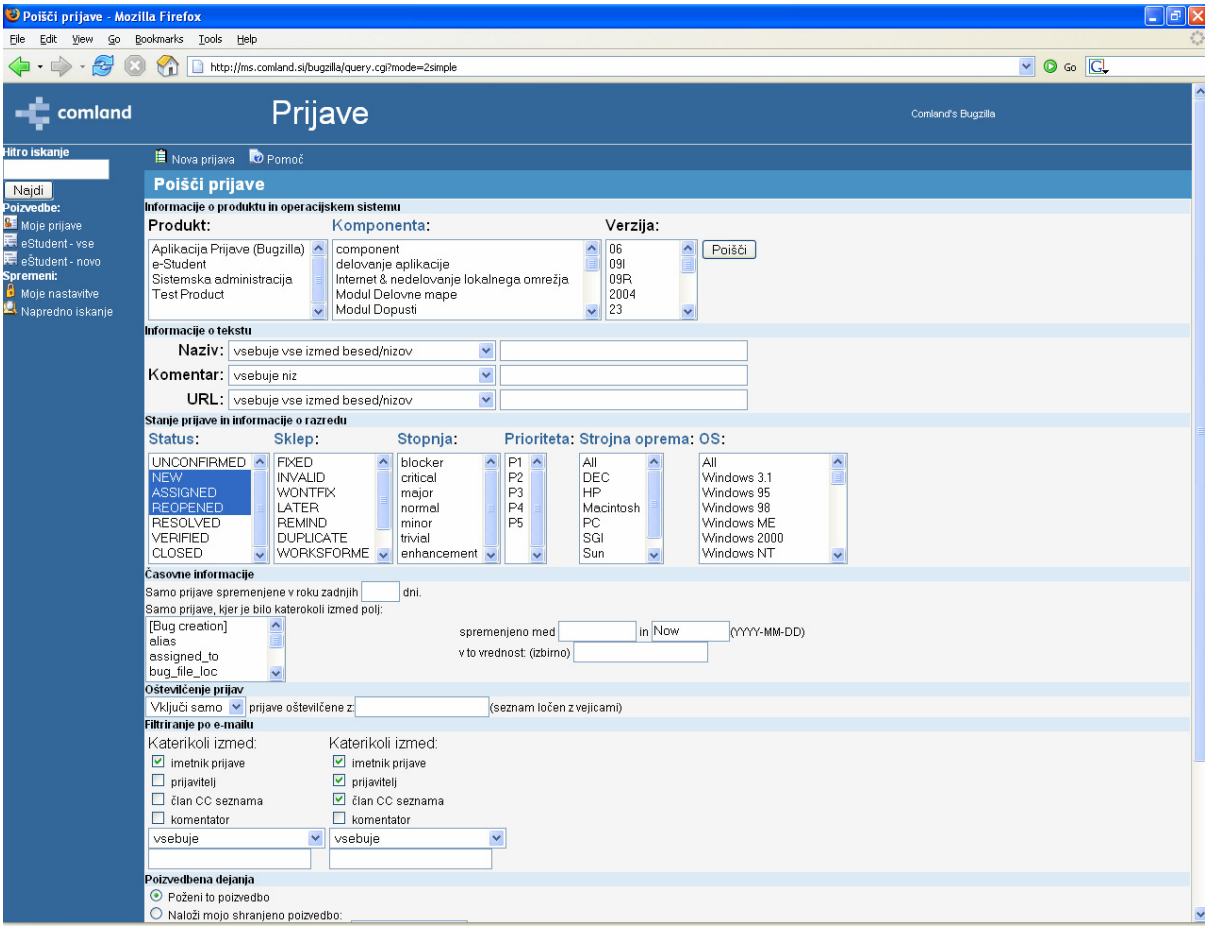

**Slika 3.19 Obrazec za prijavo v Bugzillo**




**Slika 3.20 Prva stran po prijavi v Bugzillo (pregled prijav uporabnika)**




**Slika 3.21 Vnos nove prijave**




**Slika 3.22 Napredno iskanje prijav**

### 3.4.3 Odvzemanje in dodajanje funkcionalnosti

Bugzilla je v osnovi že imela vse funkcionalnosti, ki smo jih potrebovali. Bolj natančno, ponujala nam je celo več. Zato ni bilo potrebe po kakšni drastični nadgradnji, temveč smo kvečjemu videli potrebo po izklopu določenih funkcij, ki bi utegnile za naše uporabnike biti preveč zapletene.

**Izklop glasovanja in izpisovanja pregovorov, rekov po prijavi**

Na koncu smo naredili samo dva večja, omembe vredna posega v smislu izklopa funkcionalnosti. Uporabnikom smo izklopili možnost glasovanja, to je oddajanje glasov določenim prijavam. Uporabil sem besedo izklop, ker smo funkcionalnosti dejansko le zakrili oziroma prikrili. V predlogah smo povsod, kjer so se pojavljali elementi omenjenih funkcionalnosti, le-te zakomentirali[80].

**Integracija sistema e-student in Bugzille**

Omenil sem že, da je bil eden od vzrokov, zakaj smo hitro potrebovali delujočo helpdesk aplikacijo, prevzem uporabniške podpore sistema e-student. Od Fakultete za računalništvo in informatiko (Laboratorij za informatiko) smo namreč prevzeli nadaljnje namestitve sistema e-student na fakultetah Univerze v Ljubljani. S tem smo prevzeli tudi vzdrževanje sistema in uporabniško podporo za vse že obstoječe in bodoče instance[81] e-studenta.

***Problem***

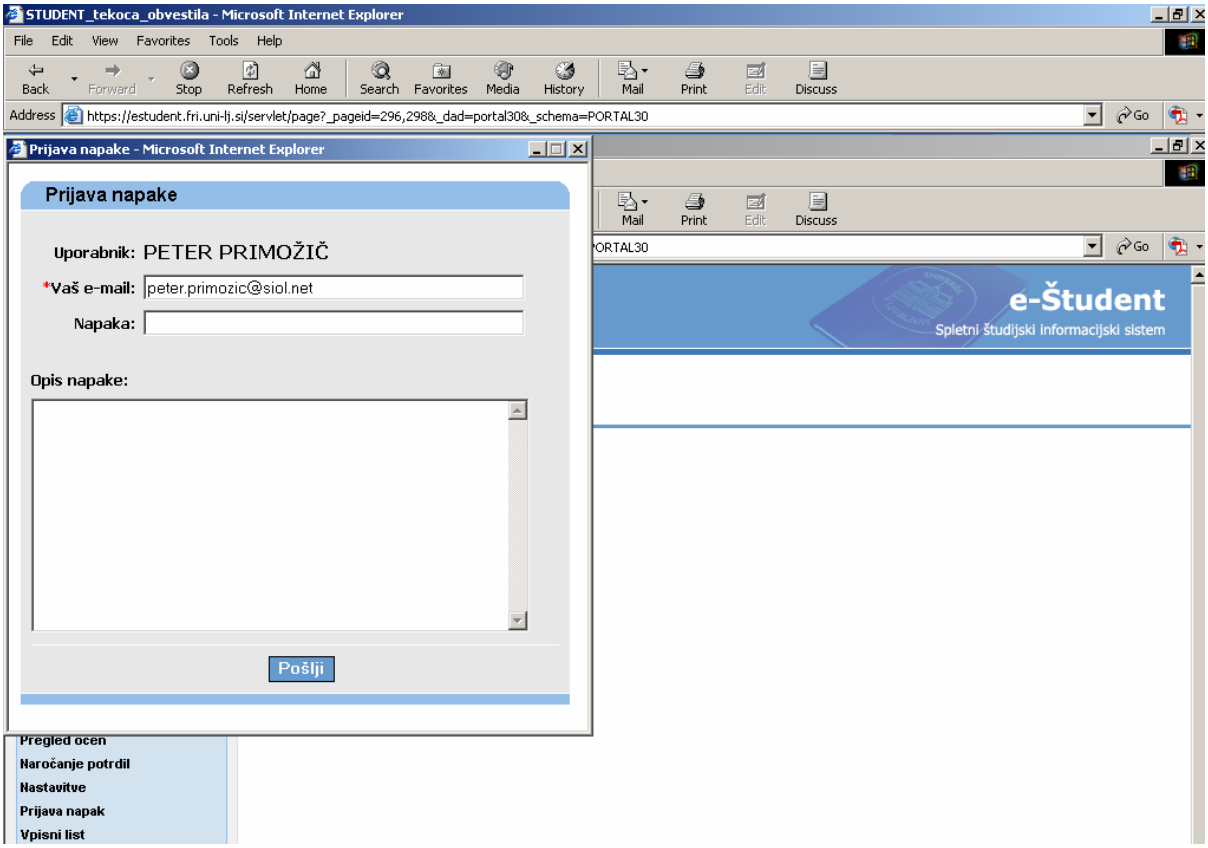
Zahtevalo se je, da lahko uporabniki sistema e-student svoje zadeve prijavljajo kar neposredno iz samega sistema. Potrebna je bila integracija sistema e-student in aplikacije Bugzilla. V osnovi Bugzilla ne omogoča take vrste komunikacije z zunanjimi aplikacijami. Zato smo morali izvesti nadgradnjo, ki bi to omogočala. Posebno težavo pri tem je predstavljalo dejstvo, da morajo imeti uporabniki za prijavo zadev v Bugzilli kreiran uporabniški račun, oziroma še več, morajo biti prijavljeni v sistem (Bugzillo). Glede na to, da je vseh uporabnikov sistema e-student nekaj deset tisoč, bi samo kreiranje njihovih uporabniških računov zahtevalo precejšen čas. Uporabniško ime v Bugzilli namreč predstavlja veljaven e-poštni naslov, posledično bi morali pridobiti več tisoč e-poštnih naslovov in narediti enolično preslikavo uporabniških imen uporabnikov v e-studentu v uporabniška imena v Bugzilli.

[80] preprečili izvajanje določenega dela programske kode
[81] posamezne namestitve sistema e-student za posamezno fakulteto

***Rešitev***

Za izvedbo integracije sta bila potrebna popravka oziroma dodatka v obeh sistemih, pri e-studentu in Bugzilli. Uporabili smo rešitev, da uporabniki preko posebnega obrazca, ki je sestavni del sistema e-student, vpišejo podatke o prijavi. Le-ti se nato s POST[82] metodo preko spletnega brskalnika posredujejo Bugzilli, ki jih ustrezno obdela. Za uresničitev ideje je bilo potrebno v sistemu e-student izdelati obrazec za pošiljanje prijav, v Bugzilli pa omogočiti sprejem prijav, posredovanih iz drugih aplikacij.

Obrazec za pošiljanje prijav v zvezi s sistemom e-student je prikazan na sliki 3.23. Poleg podatkov o uporabniku (Ime in Priimek ter poštni naslov za obveščanje o stanju prijave) smo v obrazcu zajeli tudi podatke o produktu (v tem primeru e-student) ter verziji produkta (npr.v63 predstavlja verzijo e-studenta Fakultete za Računalništvo in Informatiko).

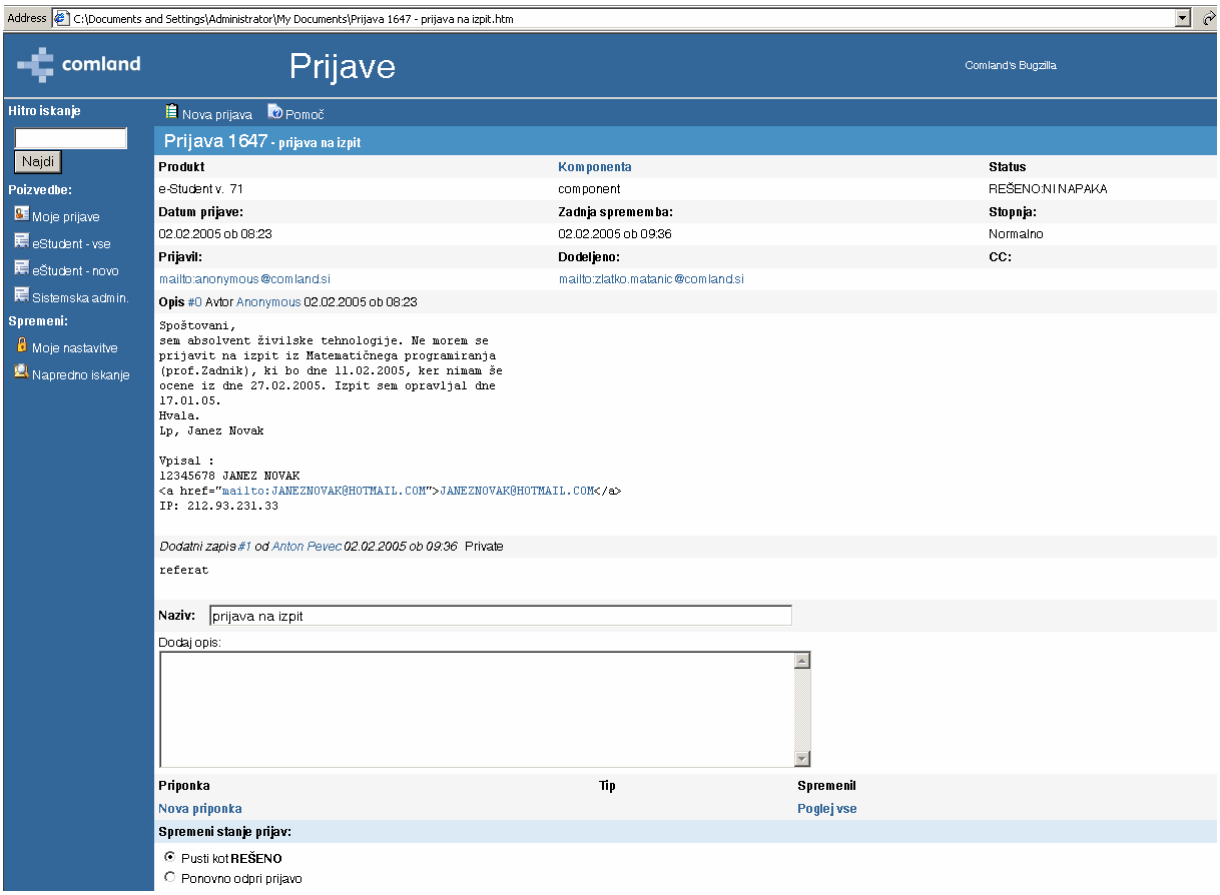



**Slika 3.23 Obrazec za prijavo zadev v sistemu e-student**

Ob kliku na gumb *Pošlji* se podatki na obrazcu s prej omenjeno POST metodo posredujejo Bugzilli oziroma njeni Perl skripti, ki podatke ustrezno obdela. Datoteko smo kreirali sami, kot temelj pa smo vzeli Bugzillino Perl skripto (*post_bug.cgi)* za pošiljanje prijave. Priredili

[82] POST metoda omogoča, da se podatki v zahtevi odjemalca pošljejo strežniku. Te podatke obdela poseben program, do katerega ima strežnik dostop (CGI skripta).

smo jo tako, da je omogočila sprejemanje prijav iz zunanjih aplikacij, pri čemer uporabnikov zunanjih aplikacij ni potrebno predhodno prijaviti v Bugzillo. Za takšne primere smo kreirali posebnega uporabnika Bugzille (uporabnika *anonymous*), ki se beleži kot nosilec vseh prijav uporabnikov zunanjih aplikacij. Naša Perl skripta iz niza znakov, ki ga sprejme, razčleni podatke o produktu in komponenti ali verziji produkta, na katerega se prijava nanaša, podatke o uporabniku, ki je prijavo poslal, ter podatke o sami vsebini prijave. Te podatke ustrezno zapiše v podatkovno bazo in posreduje obvestilo o prispeli prijavi ustreznemu skrbniku. Na sliki 3.24 je prikazan primer prijave, poslane iz sistema e-student, kot jo vidi njen skrbnik in drugi uporabniki Bugzille, ki imajo dostop do produkta e-student.

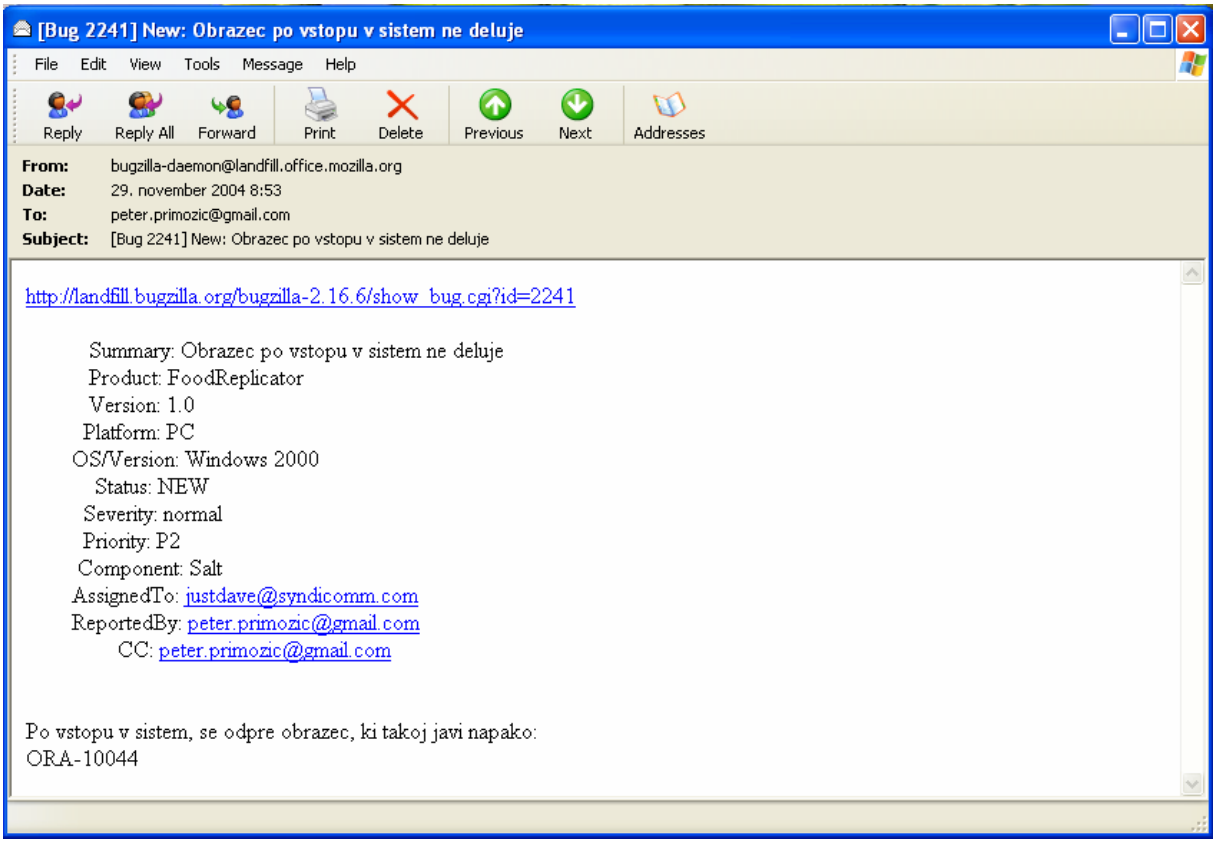



**Slika 3.24 Prikaz prijave, poslane iz sistema e-student**

Kot je razvidno na zgornji sliki, je videti kot, da je prijavo poslal *anonymous* uporabnik. Zato smo v naši Perl skripti sporočilo prijave oblikovali tako, da se na koncu sporočila izpiše, kdo je resnično podal prijavo. Tako lahko tej osebi tudi ustrezno odgovorimo.

### Nadgradnja poštnega sistema Bugzille

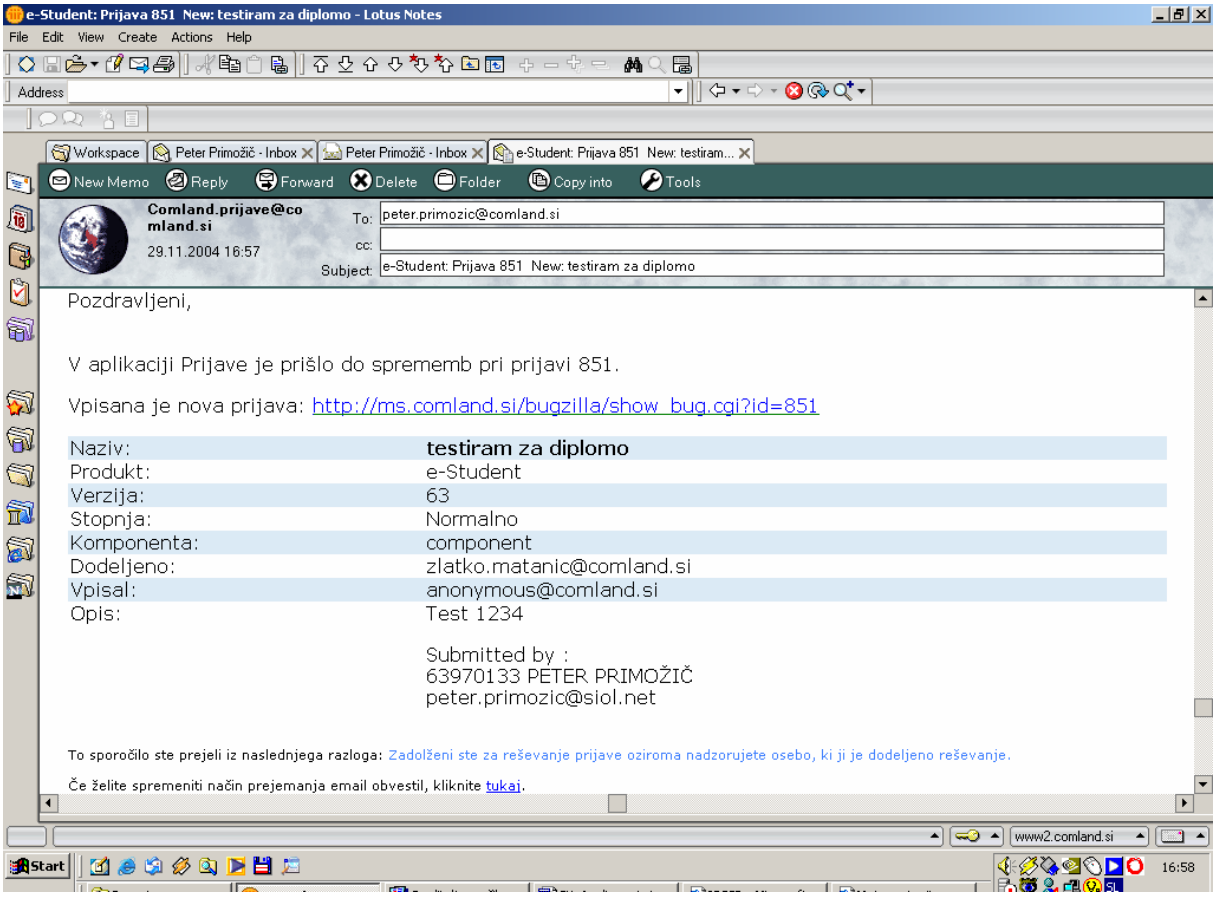
Poštni sistem (ang. mailing system) Bugzille poskrbi za obveščanje uporabnikov preko elektronske pošte o vseh spremembah in akcijah na prijavah. Uporabniki lahko v svojih nastavitvah elektronske pošte določijo, v katerih primerih želijo od Bugzille prejeti elektronsko pošto zaradi novih ali obstoječih prijav. V osnovi je izgled oziroma oblika teh elektronskih sporočil na nivoju njenega uporabniškega vmesnika, torej zelo nepregleden. Na sliki 3.25 je primer sporočila, poslanega iz osnovne (nespremenjene) Bugzille.




**Slika 3.25 Izgled elektronskega sporočila Bugzille**

Posebne potrebe po spreminjanju izgleda elektronskih sporočil ni bilo, vendar smo se tega vseeno lotili z namenom pridobiti še nekoliko prakse pri programiranju v Perl-u in hkrati bolje spoznati poštni sistem Bugzille. V datoteki *bugmail.pm*, ki poskrbi za generiranje, oblikovanje in pošiljanje elektronskih sporočil, smo dodali funkcijo, ki jih oblikuje v html obliko. Delno smo spremenili tudi izgled sporočil v tekstovni obliki. Elektronsko sporočilo, ki ga pošlje Bugzilla uporabnikom, je sestavljeno iz dveh delov, iz sporočila v tekstovni in iz identičnega sporočila v html obliki. Sporočilo se prikaže v html obliki, če ima uporabnik

odjemalca elektronske pošte, ki to omogoča, sicer se prikaže v tekstovni obliki. Na sliki 3.26 je primer elektronskega sporočila Bugzille v html obliki.

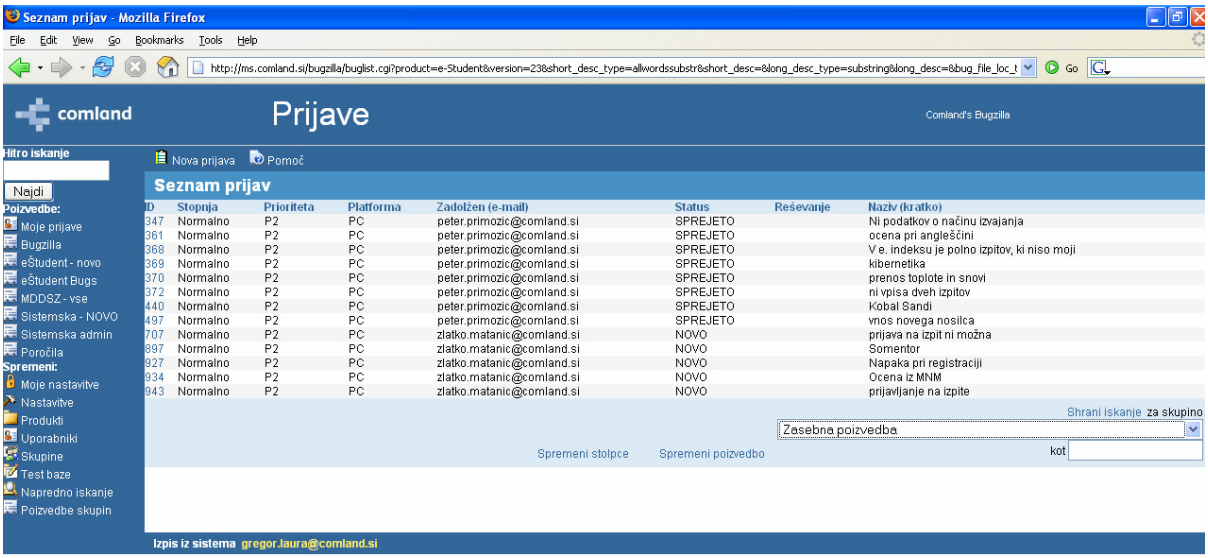



**Slika 3.26 Izgled elektronskega sporočila Bugzille v html obliki**

### Omogočenje shranjevanja poizvedb za skupino uporabnikov

Vsak uporabnik, ki v Bugzilli kreira račun, ima privzeto samo povezavo na seznam svojih prijav. Bugzilla omogoča iskanje po prijavah znotraj različnih produktov. Da uporabnikom ne bi bilo potrebno vedno znova podajati kriterijev iskanja, si lahko določeno iskalno poizvedbo shranijo. S tem se jim v meniju v levem delu strani pojavi nova povezava. S klikom na to povezavo se jim odpre seznam prijav, ki ustrezajo iskalnemu pogoju shranjene poizvedbe. Pogosto se pojavlja potreba, da bi uporabniki določene skupine poleg povezave na seznam svojih prijav, potrebovali tudi povezavo na kakšen drug seznam prijav (npr. seznam prijav vseh uporabnikov v skupini, da se prijave znotraj skupine ne bi podvajale). V osnovni Bugzilli bi moral vsak uporabnik zase shraniti poizvedbo, ki vrne željen seznam prijav, ali pa bi morali to storiti mi za vsakega uporabnika posebej. Zato smo jo (Bugzillo) tako, da omogoča shranjevanje določene poizvedbe za celotno skupino uporabnikov. To akcijo lahko

izvedejo samo uporabniki z ustreznimi (administratorskimi) pravicami. Na sliki 3.27 je prikazan seznam prijav, ki ustrezajo določeni poizvedbi. Prijavljen uporabnik ima ustrezne pravice za shranjevanje poizvedbe za skupino uporabnikov. Zato ima v spodnjem levem kotu strani spustno kombinirano polje (ang. drop-down combo box), kjer lahko izbere za katero skupino želi shraniti opravljeno poizvedbo. Če skupine ne izbere (v spustnem kombiniranem polju pusti vrednost *Zasebna poizvedba*), lahko shrani poizvedbo samo zase.




**Slika 3.27 Shranjevanje poizvedbe za skupino uporabnikov**

# 4 ZAKLJUČEK

Konkurenca na področju razvoja programske opreme je velika, zato podjetja vseskozi iščejo možnosti, ki bi jim zagotovile neko konkurenčno prednost. Menim, da odprta koda trenutno predstavlja eno takih možnosti, ki bi jo bilo vredno izkoristiti. Izkušnja podjetja Comland d.o.o. to nazorno prikazuje. Z uporabo odprtokodne aplikacije kot osnove, ki smo jo prilagodili in nadgradili lastnim potrebam, smo v podjetju prihranili približno 1000 ur v času in posledično stroških razvoja aplikacije. Največji prihranek predstavlja dejstvo, da ustrezno izbran odprtokodni produkt služi kot zelo stabilna in preizkušena osnova, ki posamezniku ali podjetju omogoča, da se osredotoči predvsem na svoje prilagoditve in nadgradnje produkta. Nekaj časa se sicer izgubi s spoznavanjem strukture in izvorne kode uporabljenega odprtokodnega programa, vendar je ta izguba minimalna. Res je, da vse odprtokodne rešitve niso tako kvalitetne kot je Bugzilla, ki smo jo uporabili mi. Vendar je kljub temu glede na številčnost odprtokodnih projektov pogosto možno najti projekt ali posamezno komponento, ki lahko zelo dobro služi kot osnova za nadaljni razvoj. Priporočeno je, da je izbran odprtokodni produkt že v stabilni ali zreli fazi razvoja, kar zagotavlja trdno in preizkušeno osnovo. Prav tako je priporočeno dobro preučiti licenco, pod katero se razširja odprtokodni program, da kasneje ne bi prišlo kršitev licenčnih pravil.

Primer uporabe odprtokodnega programa, ki sem ga predstavil v diplomski nalogi je le eden od možnih načinov za pospešitev razvoja programske opreme s pomočjo odprte kode. V našem primeru je šlo za izdelavo aplikacije, ki jo ponujamo našim naročnikom kot dodatno storitev in je ne tržimo. Pozitivna izkušnja nas je pripeljala do spoznanja, da lahko podoben pristop uporabimo tudi pri razvoju komercialnih informacijskih rešitev za naše naročnike. Zato bomo v bodoče, v kolikor nam bodo to dopuščale okoliščine, prav gotovo še velikokrat posegli po prostem programju. Poleg občutno krajšega časa razvoja prinaša prosto programje razvijalcem in podjetju nova spoznanja, ideje in znanja, ki se skrivajo v njem. Mnogi se uporabi prostega programja pri razvoju programske opreme izogibajo, ker ne želijo posredovati svoje izvorne kode (poslovnih skrivnosti). Menim, da je bojazen pred posredovanjem izvorne kode odveč. Dejstvo je, da večina uporabnikov sploh ne ve, kaj je izvorna koda in kaj bi z njo počeli. Poleg tega jo je podjetje oziroma posameznik je primoran posredovati le v primeru, da uporabnik to od njega zahteva.

Na splošno sem mnenja, da možnosti, ki jih ponuja prosto programje razvijalcem in podjetjem niso dovolj dobro raziskane. Prav tako je na področju poslovnih modelov, ki se razvijajo okrog prostega programja. Menim, da prosto programje ne bo nikoli nadomestilo zaprtega (lastniškega) programja – ljudje smo po naravi enostavno preveč pohlepni za to. Na koncu bo verjetno prišlo do oblikovanja ravno pravšnjega razmerja med pol-zaprtimi in odprtimi sistemi.

Upam, da sem v svoji diplomski nalogi uspel dovolj nazorno predstaviti enega od načinov uporabe odprte kode pri razvoju programske opreme. Svojim kolegom pa prepuščam, da raziščejo in predstavijo še katerega od drugih načinov razvoja z odprto kodo.

# ZAHVALA

Iskreno se zahvaljujem mentorju, prof. dr. Franc Solini, univ. dipl. ing., za upoštevanje mojih zamisli in želja ter njegovo strokovno pomoč in vodenje pri opravljanju diplomske naloge. Zahvala gre tudi sodelavcu Gregorju Lauri za vse koristne informacije ter Aleksandru Lazareviću za njegove nasvete.

Še posebej se zahvaljujem Maji Lubarda za izredno pomoč pri dokončnem oblikovanju in lektoriranju diplomske naloge ter njeno spodbudo in podporo, ki sem je bil deležen ves čas pisanja naloge.

Za vso podporo, potrpljenje in razumevanje pa bi se rad srčno zahvalil tudi svojim staršem in sestri.

# VIRI IN LITERATURA


[1] OSI - Open Source Initiative, [http://www.opensource.org/index.php]

[2] OSI - Open Source Definition, [http://www.opensource.org/docs/definition.php]

[3] Free Software Foundation: GNU General Public Licence [http://www.fsf.org/licenses/gpl.html], Junij 1991

[4] Z. Barović, Razumeti odprto kodo, Revija Moj mikro, April 2004

[5] M. S. Elliot: The virtual organizational culture of a Free software development community

[6] G. von Krogh, S. Haefliger, S. Spaeth, Collective Action and Communal Resources in Open Source Software Development: The Case of Freenet [http://opensource.mit.edu/papers/vonkroghhaefligerspaeth.pdf]

[7] M. Damjan: Pravni temelj prostega programja, revija Pravnik, let.59 (2004), 4-6

[8] E. Moglen, Free Software Matters: Free Software or Open Source? [http://emoglen.law.columbia.edu/publications/lu-07.html]

[9] G. J. Rothfuss: A framework for open source projects [http://opensource.mit.edu/papers/rotfuss.pdf]

[10] E. E. Kim, An introduction to open source communities, April 2003 [http://opensource.mit.edu/papers/blueoxen.pdf]

[11] A. Nuvolari: Open source software development: Some historical perspectives [http://opensource.mit.edu/papers/nuvolari.pdf]

[12] R. M. Stallman, "The GNU Operating System and the Free SoftwareMovement", v DiBona C., Ockman S. in Stone M. (eds.), *Open Sources: Voices from the Open Source Revolution,* O'Reilly: Sebastopol, California

[13] LEF, Open Source: Open For Business, [http://www.csc.com/features/2004/uploads/LEF_OpenSource_ExecutiveSummary.pdf]

[14] Lakhani, R. Karim., B. Wolf, J. Bates in C. DiBona, The Boston Consulting Group Hacker Survey, 24.7.2004, [http://www.osdn.com/bcg/BCGHACKERSURVEY-0.73.pdf]

[15] Institute of Infonomics, University of Maastricht, Nizozemska in Berlecon Research GmbH, Berlin, Nemčija, Free/Libre and Open Source Software: Survey and Study Final Report, Junij 2002, [http://www.infonomics.nl/FLOSS/report/]

[16] Technical University of Berlin, WIDI – Who Is Doing It? 14.8.2001, [http://widi.berlios.de/]

[17] Razdevšek-Pučko, Teze predavanj, Motivacija in učenje, Pedagoška Fakulteta, Univerza v Ljubljani

[18] A. Woolfolk, Pedagoška psihologija, Educy, 2002, str.165–166, 317–383,441-445

[19] B. Marentič-Požarnik, Psihologija učenja in pouka, DZS, 2000 183-219, str.259-262, 232-233

[20] Motivacija [http://www2.arnes.si/~skunci1/motivacija.htm]

[21] A. Hemetsberger, Sharing and creating knowledge in open-source communitites, The case of KDE, 2004 [http://opensource.mit.edu/papers/hemreinh.pdf]

[22] R. Teigland, Communities of Practice in a High-Technology Firm, v Birkinshaw in Hagstrom (Eds) The Flexible Firm: Capability Management in Network Organizations, Oxford University Press, New York, 2000

[23] T. R. Madanmohan, S. Navelkar, Roles and Knowledge Management in Online Technology Communities: An etnography study, [http://opensource.mit.edu/papers/madanmohan2.pdf]

[24] M. O'Brien, Ask Mr. Protocol, The Cathedral, the Bazaar and Mr. P, Revija SunExpert Magazine, April 1998, [http://swexpert.com/C1/SE.C1.APR.98.pdf]

[25] G. Polančič, R. V. Horvat, Prednosti in slabosti uporabe odprto-kodnih ogrodij, 2004 [http://lisa.uni-mb.si/%7Epolancic/si/raziskovalnoDelo/publikacije/Prednosti_in_slabosti_uporabe_odprtokodnih_ogrodij.pdf]

[26] R. P. Gabriel, R. Goldman, Open Source: Beyond the Fairytales, 2002 [http://opensource.mit.edu/papers/gabrielgoldman.pdf]

[27] G. Polančič, Kakovost programske opreme tipa »odprta koda«, Individualno raziskovalno delo

[28] Benefits of using Open Source [http://open-source.gbdirect.co.uk/migration/benefit.html#reliability]

[29] C. A. Kenwood, A Business Case Study of Open Source Software, Julij 2001 [http://www.mitre.org/work/tech_papers/tech_papers_01/kenwood_software/kenwood_software.pdf]

[30] Dobičkonosna odprta koda, april 2003 [http://www.lugos.si/arhiv/prispevki/dok]

[31] The Free Sofware Definition [http://www.fsf.org/philosophy/free-sw.html]

[32] J. Benčina, V pričakovanju hitre rasti uveljavljanja odprte kode, INDO 2003: Posvetovanje informatikov v državni upravi, Zbornik referatov, september 2003, str. 456-458

# Izjava o samostojnosti dela

Izjavljam, da sem diplomsko delo izdelal samostojno pod vodstvom mentorja prof. dr. Franca Soline, univ. dipl. ing. Izkazano pomoč drugih sodelavcev sem navedel v zahvali.

*Peter Primožič*